1N-60
85248
122P.

JPL PUBLICATION 87-18

Hypercube Matrix Computation Task

Final Report for 1986 – 87

R. Calalo
W. Imbriale
P. Liewer
J. Lyons
F. Manshadi
J. Patterson

```
(NASA-CR-180921)   HYPERCUBE MATRIX                    N87-25772
COMPUTATION TASK Final Report, 1986-1987
(Jet Propulsion Lab.)   122 p   Avail: NTIS
HC   A06/MF   A01                      CSCL 09B        Unclas
                                               G3/60   0085248
```

July 1, 1987

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL PUBLICATION 87-18

Hypercube Matrix Computation Task

Final Report for 1986 – 87

R. Calalo
W. Imbriale
P. Liewer
J. Lyons
F. Manshadi
J. Patterson

July 1, 1987

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

ABSTRACT

The objective of the Hypercube Matrix Computation (Year 1986-1987) task was to investigate the applicability of a parallel computing architecture to the solution of large scale electromagnetic scattering problems. Two existing electromagnetic scattering codes were selected for conversion to the Mark III Hypercube concurrent computing environment. These two codes were selected so that the underlying numerical algorithms utilized would be different thereby providing a more thorough evaluation of the appropriateness of the parallel environment for these types of problems. The first code was a frequency domain method of moments solution, NEC-2, developed at Lawrence Livermore National Laboratory. The second code was a time domain finite difference solution of Maxwell's equations to solve for the scattered fields.

Once the codes were implemented on the hypercube and verified to obtain correct solutions by comparing the results with those from sequential runs, several measures were used to evaluate the performance of the two codes. First, a comparison was provided of the problem size possible on the hypercube with 128 megabytes of memory for a 32-node configuration with that available in a typical sequential user environment of 4 to 8 megabytes. Then, the performance of the codes was analyzed for the computational speedup attained by the parallel architecture. The speedup can be measured a) by comparing the CPU times for key components of the code running on the 32-node Mark III Hypercube with the CPU times for the same code components running on a VAX 11/750, b) by comparing the times for the code running in 32 nodes relative to the code running in a single node, and c) by comparing the times (and therefore scalability to larger hypercube configurations) when the problem size per node is fixed and the number of nodes in use is varied.

CONTENTS

PRECEDING PAGE BLANK NOT FILMED

Figures

Tables

SECTION I

INTRODUCTION

A. THE OBJECTIVES

The basic objective of the task was to investigate the applicability of a parallel computing architecture to the solution of large scale electromagnetic scattering problems. Specifically, the task was to implement two electromagnetic scattering codes encompassing different numerical algorithms on the Mark III Hypercube. The two selected algorithms utilized 1) a frequency domain computer code which implemented the method of moments solution, and 2) a time domain code which used a finite difference solution of Maxwell's equations to solve for the scattered fields. Important measures for demonstrating the utility of the parallel architecture were the size of the problem that could be solved and the efficiency by which the paralleling could increase the speed of execution. In particular, the results of the parallel codes would be compared for both accuracy and speed with the sequential version of the code running on a standard mainframe.

B. THE MARK III HYPERCUBE

Recent advances in high speed microprocessor technology and in methods to combine large numbers of these processors into concurrent structures introduce cost-effective means for solving massive computing problems. A number of different schemes for connectivity as well as for the components of the computing elements for such concurrent architectures have been suggested. Some machines utilize massive numbers of small-grain computing elements; others concentrate large-grain computing capability in a modest number of distributed computing elements. One such architecture is the California Institute of Technology (Caltech)/Jet Propulsion Laboratory (JPL) Hypercube. A hypercube is a connectivity scheme which can be viewed as an array of N processors where each is capable of communicating directly with $n = \log_2 N$ neighboring processors along the edges of an n-dimensional cube. The Caltech/JPL Hypercube is now in its third generation of development. At this time configurations consist of up to 32 nodes. A 128-node Mark III is now under construction. Although several commercial concerns produce hypercube architecture concurrent computers, the Mark III offers the state of the art computing capability for such machines.

The Mark III Hypercube node has a pair of Motorola 68020 processors--one is the main application processor and the second is the communication processor (Figure 1.1). The communication processor handles internode communication generally without the need to interrupt the main processor. Currently the hypercube uses the Motorola 68881 floating point co-processor which delivers 60 to 120 thousand floating point operations per second (kiloflops) per node. A new floating point daughterboard is under development which will boost this performance to 5 to 10 megaflops per node. Each node has 4 megabytes of dynamic memory and 128 kilobytes of static memory. Communications flow at 2 megabytes per second per channel with a node capable of communicating on all of its channels simultaneously.

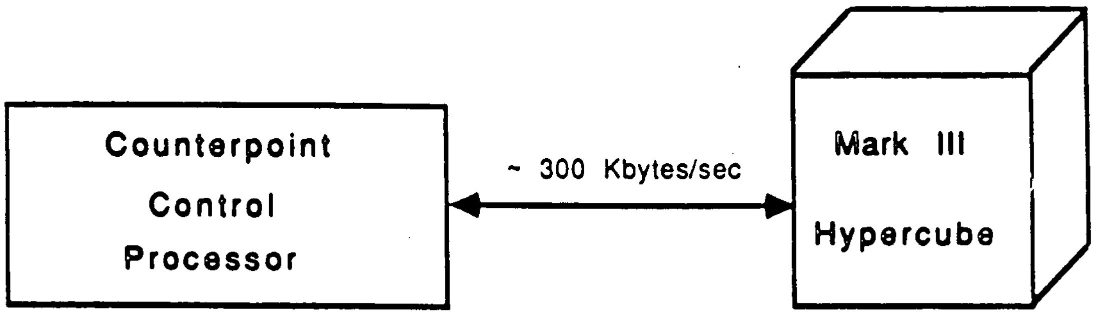
Each node:

2 Motorola 68020 processors
- main cpu
- I/O processor

Motorola 68881 floating point processor

4 MBytes dynamic RAM

128 KBytes static RAM



Operating systems:
Crystalline
Mercury (asynchronous)

Languages supported:
C
FORTRAN

Interprocessor communication rate:

2 MBytes/sec

Figure 1.1 The Mark III Hypercube System.

Two operating systems have been developed for the Mark III Hypercube. The first is the synchronous or Crystalline Operating System (CrOS). In this communicating regime nodes run asynchronously, aligning activity only when communication is required between two processors. For some applications the message passing requirements are irregular. For this reason a second operating system known as Mercury was developed to handle asynchronous message traffic. An application can flow from one operating system to the other within a single code. The Mark III Hypercube supports both C and FORTRAN programming languages.

SECTION II

TASK DESCRIPTION

A. OBJECTIVES

The objectives of the Matrix Computation Task were to convert two electromagnetic scattering codes to the hypercube parallel computing environment and to analyze the results as to correctness and performance. These two codes differ in the underlying techniques used for obtaining electromagnetic scattering solutions. The one code is a frequency domain based code while the other is time domain.

B. TECHNICAL APPROACH

The first phase of the task was to select the codes which would be used for the conversions. Since both codes were either implemented on or portable to a VAX, the next step was to run the codes on a VAX. Eventually the VAX runs would be used to benchmark the performance of the respective codes. The next phase was to transport the codes to the Counterpoint System XIX computer environment. The Counterpoint which is a Motorola 68020 based system serves as the host to the Mark III Hypercube. Once this phase was completed, then began the actual conversion of the two codes to the parallel computer.

The task team has six members, three of which worked on the time domain code and three on the frequency domain. Two teams were selected rather than one team doing both codes back to back to reduce the time span of the contract. A more detailed description of the task phases follows.

1. Selection of Codes

The selection of the appropriate frequency domain code for the parallel conversion was determined from the onset. The widely used Numerical Electromagnetic (Method of Moments) Code (NEC-2) developed at Lawrence Livermore Laboratories is the premier code for frequency domain solutions and as such it is well-documented and examples of solutions are readily available.

The selection of the appropriate time domain code was not as simple. Therefore, the first phase of the task included a survey of existing time domain codes. The K.S. Kunz' Generalized Finite Difference Time Domain Code GFDTD, although limited in scope, was determined to provide an adequate basic code. During the course of the task, this code was enhanced to provide additional capabilities.

2. Transferring to the VAX

Both codes transferred easily to the VAX environment. For the NEC code possible future portability problems were anticipated. NEC is written in FORTRAN-77 utilizing VAX extensions including system calls and complex data types. Because the Counterpoint adheres strictly to the FORTRAN-77 and hence does not include the complex data type, it was decided to convert the complex data variables to arrays and provide the complex functions as subroutines first on the VAX. In this way the validity of any modifications could be checked before moving the code to the Counterpoint. Because most of the variables within the NEC code are of the complex data type, significant time was required to perform these changes.

3. Transferring to the Counterpoint

Once the codes were verified on the VAX, they each were transferred to the Counterpoint. Again, comparisons of the results were performed. Differences in precision for single and double precision of the VAX and Counterpoint were eliminated as a source of disparity between solutions; excellent agreement was found between results on the two computers for both codes.

4. Parallel Code Conversion

The most significant portion of the time on the task was the actual conversion to a parallel code. The finite difference time domain code team first implemented the GFDTD code. The correctness of the parallel solutions were verified by comparisons with the results of sequential runs both on the VAX and on the Counterpoint. The results were also compared with a similar code developed by Allen Taflove. The results from runs of the two codes did not demonstrate as good an agreement as was expected. Taflove had validated his results by comparing his results with the method of moment ones. It was determined that the parallel finite difference code should be enhanced to include second order radiation boundary conditions. The second order conditions suppress waves impinging on the truncation planes at angles as large as 45 degrees. The fact that the first order conditions have a fundamental limit of small angles of incidence was expected to be the major reason for the disparity between results. A second comparison of results then produced good agreement.

The method of moments frequency domain code was implemented in stages. The first capability to be incorporated was the wire model. In order to have a sequential code to use for comparison, a modified sequential code was developed extracting only those elements from the NEC code which were necessary for the solution of the wire case. This subset of NEC was called "Short-NEC". Included in the preparation of Short-NEC was the transformation of the code from a transposed fill of the interaction matrix to a "normal" (i.e. not transposed) fill. Once the sequential version of Short-NEC was completed and tested, work began on the parallel version. One by one routines were converted to run on the hypercube rather than on the Counterpoint. Once the wire capability had been tested, patch, extended thin wires, and loading were added to form the version which was called "Medium-NEC". Along with the extended capabilities, the ability to handle symmetric cases and multiple right-hand side excitation vectors was provided. The sequential LU decomposition factorization and solution were replaced by the parallel Householder transformation and solution.

5. Comparison of Results

As has been noted, at all stages of the code development comparisons were done to assure the correctness of the obtained solution. In the case of the finite difference code, there no longer was a sequential code to use for verification. Therefore, the Taflove code was used for this comparison.

6. Analysis of Performance

Timing comparisons were run between the VAX codes and their parallel counterparts. Also timing comparisons were obtained for the code running on the Counterpoint and one node of the hypercube. Again, since no sequential code exists which incorporates the full capability of the enhanced parallel finite difference code, the timing was obtained by comparing the results of the multiple node hypercube with those of the code running on one node.

7. The Sponsor-Selected Test Case

The sponsor representative specified a particular test case to be run on both codes. The case is a plane wave incident on a sphere. The modelling takes no advantage of the properties of symmetry of the object. The results are compared using the radar cross section (RCS) of the two solutions. In addition, the degree of agreement with the exact solution is analyzed.

SECTION III

TIME DOMAIN CODE

A. DESCRIPTION

The Finite-Difference Time-Domain (FDTD) method employs an iterative solution to Maxwell's time-dependent curl equations to determine the fields associated with complicated scattering objects. Because of the simplicity and directness of the method, the required memory and execution times increase only linearly with the number of cells in the computation region. Hence, FDTD is particularly well suited to electromagnetic analysis of structures with volumetric complexity (e.g., inhomogeneous dielectrics).

Since FDTD is a time-domain code, it most naturally lends itself to problems involving transient effects. K. Yee's [3-1] pioneering work on FDTD considered such applications. The application of the method to the determination of steady-state fields, and in particular to the calculation of radar cross sections (RCS), has been pioneered by A. Taflove and colleagues [3-2] at Northwestern University. It is the application of FDTD to RCS computation that is relevant to this report.

Taflove provided us with a copy of his FDTD FORTRAN code along with a report [3-3] describing the method and its application to RCS studies. Initially, we planned to decompose this code into parallel form for the hypercube, but the monolithic nature of the code made this impractical. Instead we obtained a FDTD FORTRAN code (GFDTD) and user's manual [3-4] from Lawrence Livermore National Laboratory (LLNL). (Note that [3-4] is actually not for the exact code that we received.) The modular form and straightforward coding of GFDTD greatly eased the task of parallel decomposition.

However, the GFDTD code was designed for the analysis of transient currents. We added (in parallel) the features required to make an RCS code with capabilities similar to Taflove's code. In making these additions we closely followed the approach taken by Taflove. A good discussion of this approach is given in [3-3].

Note that there is one major difference between our RCS code and Taflove's RCS code: Taflove employs the total-field form of the curl equations, whereas we (and LLNL) use the scattered-field curl equations. According to Taflove [3-3] and [3-5], the total-field formulation more accurately determines the total-fields in shadow regions than does the scattered-field formulation. However, since we are only interested in the scattered-fields in this work, Taflove's argument does not apply.

Below we present an outline of the theory and algorithms comprising our code. This is supported by a flowchart of the code, Figure 3.1.

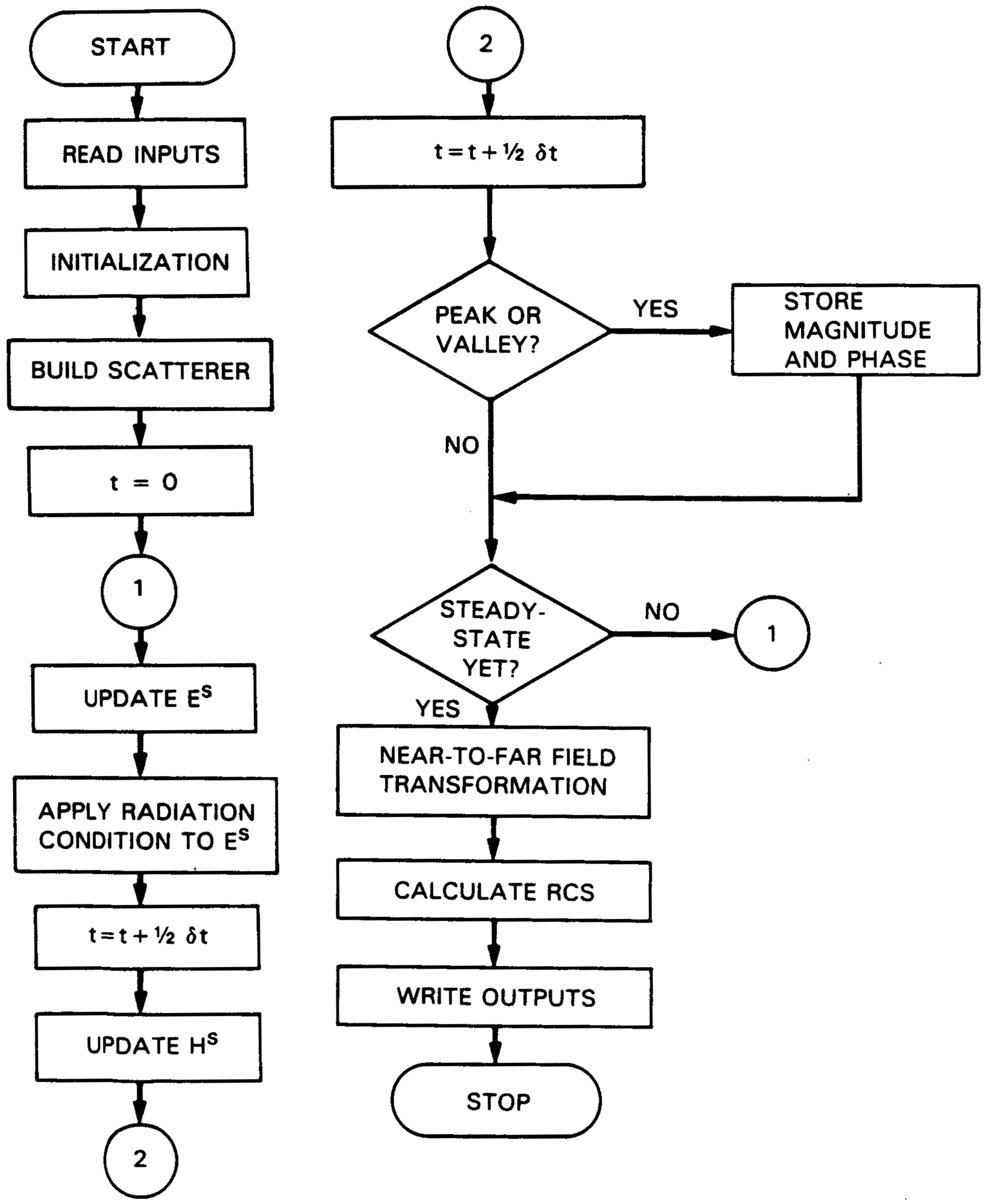

START
READ INPUTS
INITIALIZATION
BUILD SCATTERER
t = 0
1
UPDATE E^S
APPLY RADIATION CONDITION TO E^S
$t = t + \frac{1}{2}\,\delta t$
UPDATE H^S
2
2
$t = t + \frac{1}{2}\,\delta t$
PEAK OR VALLEY?
YES
STORE MAGNITUDE AND PHASE
NO
STEADY-STATE YET?
NO
1
YES
NEAR-TO-FAR FIELD TRANSFORMATION
CALCULATE RCS
WRITE OUTPUTS
STOP

Figure 3.1 Block diagram of FDTD algorithm.

1. The Finite-Difference Time-Domain Method

The FDTD method is a direct solution of Maxwell's time-dependent curl equations. The goal is to model the propagation of an EM wave into a volume of space containing a dielectric or conducting structure. By time-stepping, i.e., repeatedly implementing a finite-difference analog of the curl equations at each cell of the corresponding space lattice, the incident wave is tracked as it first propagates to and interacts with the scattering object. Wave-tracking is completed when the desired late-time or sinusoidal steady-state behavior is observed at each lattice cell. This procedure, while computationally intensive, achieves simplification by analyzing the interaction of the wavefront with portions of the structure surface at a given instant in time, rather than attempting a simultaneous solution of the entire problem.

Modeling of the scattering object is achieved by embedding the object in a lattice composed of FDTD unit cells as shown in Figure 3.2. The arrangement of the electric and magnetic field components about the unit cell is shown in the upper left of the figure. This arrangement and the corresponding finite-difference formulation were first proposed by Yee [3-1].

The structure of interest is mapped into the lattice by first choosing the space increment and then assigning values of permittivity and conductivity (and/or permeability and magnetic loss) to each component of E (and/or H). No special handling of EM boundary conditions at media interfaces is required because the Yee formulation of the curl equations satisfies these conditions automatically; hence, the basic computer program need not be modified to change from structure to structure. Inhomogeneities and fine details can be modeled with a maximum resolution of one unit cell.

The curl equations are cast into the form

$$\mu \frac{\partial H^s}{\partial t} = -\nabla \times \bar{E}^s - (\mu - \mu o) \frac{\partial \bar{H}^i}{\partial t} \qquad (3.A.1a)$$

$$\varepsilon \frac{\partial E^s}{\partial t} + \sigma \bar{E}^s = \nabla \times \bar{H}^s - \sigma \bar{E}^i - (\varepsilon - \varepsilon o) \frac{\partial \bar{E}^i}{\partial t} \qquad (3.A.1b)$$

This formulation is termed the "scattered-field" formulation, where the scattered-fields are defined to be the total-fields minus the incident-fields. Note that the total-fields are the true fields in the presence of the scatterer, while the incident fields are defined to exist in empty space. In empty space, equations (3.A.1) reduce to the form

$$\mu_o \frac{\partial \bar{H}^s}{\partial t} = - \nabla \times \bar{E}^s \qquad (3.A.2a)$$

$$\varepsilon_o \frac{\partial \bar{E}^s}{\partial t} = \nabla \times \bar{H}^s \qquad (3.A.2b)$$

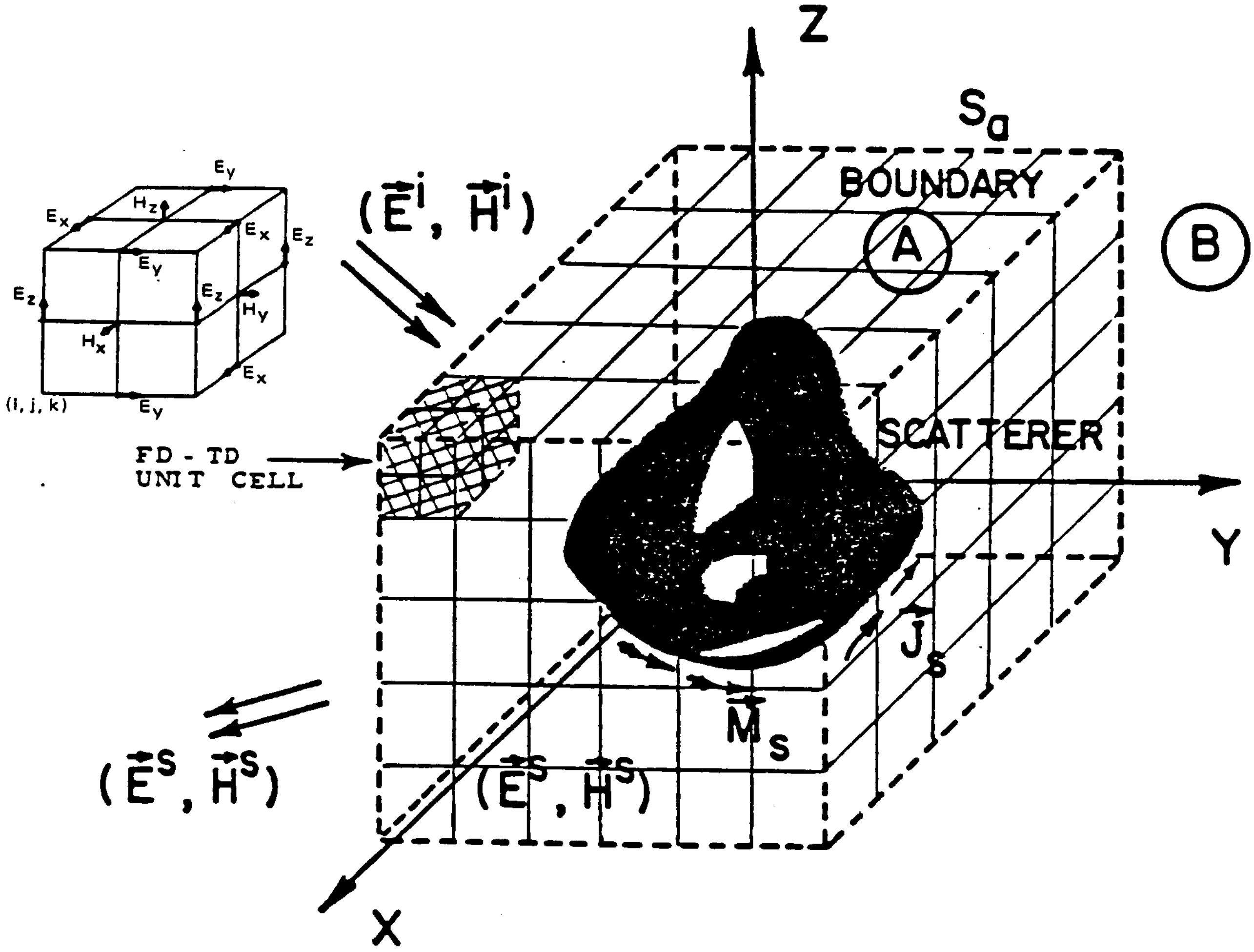


Figure 3.2 Scattering object embedded in a FDTD lattice.
(This figure is reproduced, courtesy of the authors of Ref. [3-3].)

For the case of a perfectly conducting medium, equation (3.A.1a) reduces to equation (3.A.2a), but equation (3.A.1b) takes the form of the boundary condition

$$\bar{E}^{s}_{tan} = -\bar{E}^{i}_{tan} \tag{3.A.3}$$

Equations (3.A.1), (3.A.2) and (3.A.3) are converted to finite-difference form by first expressing the vector equations as separate scalar equations in rectangular coordinates. Next, two-point central differencing is employed in both spatial and temporal coordinates to discretize the scalar equations. With the Yee formulation, second-order accuracy in the space and time increments is achieved. The mathematical details are discussed in [3-3] and [3-4].

To obtain accurate scattered fields, the size of the unit cell (i.e., the spatial finite-difference step) must be a small fraction of the electrical size of the scatterer. Taflove [3-3] claims to obtain a near field accuracy of ±7% for $\delta = \lambda/10$, and ±3% for $\delta = \lambda/20$, where 'δ' is the side of a cubic unit cell.

Once a unit cell size is specified, the time-step (i.e., temporal finite-difference step) is determined from the requirement of algorithm stability. The so-called Courant stability condition requires a time-step δt which satisfies

$$\delta t \leq \frac{\delta}{\sqrt{3} \cdot c} \tag{3.A.4}$$

where 'c' is the speed of light. A derivation of this equation is given in an appendix of [3-2].

The FDTD algorithm evaluates $\bar{E}^s$ and $\bar{H}^s$ at alternate half time-steps in a manner directly analogous to the solenoidal generation of fields as expressed by the curl equations. Specifically, the components of E^s at time-step n + 1/2 (integer n) are determined from the appropriate components of H^s at time-step n and from their own values at time-step n - 1/2. Similarly, the components of H_s at time-step n + 1 are determined from the appropriate components of E^s at time-step n + 1/2 and from their own values at time-step n. In this manner, updated values of E_s and H_s are computed during each full time-step for every point in the lattice.

At the lattice boundary this algorithm can no longer be used to update field values, because points outside the lattice would be needed. Hence, a radiation condition is applied to the scattered fields at each lattice truncation plane to simulate the extension of the lattice to infinity. Specifically, the radiation condition gives us a formula by which the components of E_s lying in a given truncation plane are updated using their own values at the previous time-step. Formulas for first-order and second-order (i.e., containing first and second derivatives in space and time) radiation conditions and their corresponding central-difference expressions are given in [3-6]. A discussion of the derivation of radiation conditions from the scalar wave equation may be found in [3-7].

The sole reason for employing high-order radiation conditions is that their greater accuracy allows us to truncate the lattice closer to the scattering object, thus saving storage and computation. The first-order radiation condition yields accurate near fields only for lattice dimensions greater than twice the dimension of the scatterer. The second-order radiation condition, while considerably more complicated, allows us to use a lattice less than twice the dimension of the scatterer.

As mentioned above, we generated a parallel FDTD code similar to Taflove's sequential code by starting with a LLNL FDTD code (GFDTD). Because the GFDTD was used for the analysis of transient currents, extensive modifications were necessary to make a code capable of RCS calculations. These modifications in order of implementation are the following:

(1) plane wave incident field

(2) second-order radiation condition

(3) magnitude and phase calculation for sinusoidal steady-state

(4) near-to-far field transformation.

Modifications (1) and (2) represent changes to already existing features of GFDTD. The original code had an exponential pulse-type incident field and employed the first-order radiation-condition. We modified the code to have a uniform, monochromatic, plane wave incident field, and to use the second-order radiation condition. The latter involved extensive code changes.

Standard RCS computation is performed with steady-state far fields, whereas FDTD computes the time-evolution of the near fields. Thus, modifications (3) and (4) represent additions to GFDTD.

To achieve steady-state with FDTD, the algorithm is marched-in-time through several cycles of the incident field. The peaks and valleys of the sinusoidally varying waveform are monitored by computing the first and second time-derivatives at various points in the lattice. At a peak or valley, the first derivative is zero. The second derivative is used to determine specifically which extremum is present. After several cycles of the incident field, the peaks and valleys converge and the fields have reached steady-state. The fields at the monitored lattice points may then be expressed as a magnitude and phase. The magnitude is computed as one half the difference between the peak and valley amplitudes. The phase is found by first recording the time-step number at which a peak occurs and subtracting it from a reference time-step number. The resulting time difference is then converted to radians by multiplication with the incident field angular frequency. Note that the reference time-step is an arbitrarily chosen time which occurs before the start of steady-state and which must remain a constant for the entire run.

The specific points in the lattice for which steady-state magnitude and phase are computed are determined by the near-to-far field transformation. This transformation relates the tangential E and H fields over a closed surface to the fields at any point outside the volume enclosed by the surface. We define this surface to be the "integration surface", and choose it to be the surface existing several cells inside the boundary of the computation lattice. After steady-state has been reached and at the end of time-stepping, the magnitude and phase for each point on the integration surface are stored. These points are then integrated (with an appropriate weighting-function) over the integration surface to determine the far fields in a particular direction.

Implementing the magnitude and phase computation and the near-to-far field transformation required significant additional coding. Computing the RCS, once the appropriate far-fields have been obtained, is a straightforward matter.

The mathematical details of the near-to-far field transformation, as well as a more in-depth discussion of the second-order radiation condition and the magnitude and phase computation, may be found in [3-3].

2. Block Diagram of Algorithm

A block diagram outlining the sequence of operations in the FDTD algorithm is shown in Figure 3.1. In general, a subroutine (or several subroutines) of FORTRAN code corresponds to each block. This block diagram is specifically for the code generated in this work (as outlined in Section 3.A.1), but does <u>not</u> describe the parallel decomposition of the code.

B. PARALLEL DECOMPOSITION

The Generalized Finite Difference Time Domain (GFDTD) code, explained in detail below, requires discrete field components at a distance of at most one half a unit cell to update a particular field component within the computation lattice. GFDTD requires discrete field components at a distance of at most two unit cells to update a particular field component on the truncation planes. When GFDTD calculates Radar Cross Sections (RCS), portions of the integration planes, which we use to evaluate vector potentials, reside in distinct nodes of a Hypercube. Because of these features, which include field updates using small volumes of the computation lattice and RCS calculations using contributions from distinct areas of the computation lattice, we can construct a Parallel Finite Difference Time Domain (PFDTD) code.

1. Generalized Finite Difference Time Domain Code

In this section, we give a detailed explanation of the Generalized Finite Difference Time Domain (GFDTD) code for solving electromagnetic scattering problems. In the next section, we elaborate on the Parallel Finite Difference Time Domain (PFDTD) code.

GFDTD uses a finite difference approximation to Maxwell's curl equations to solve electromagnetic scattering problems. Refer to Figures 3.3 and 3.4 for Maxwell's equations in finite difference form. Figure 3.3 shows the finite difference equation used to update the x component of the electric field at the current time, n. Figure 3.4 shows the finite difference equation used to update the x component of the magnetic field at the current time, n. We explain the details of these updates below. There are similar equations for the update of the other four field components.

According to the Yee method of updating discrete scattered electric and magnetic field components, GFDTD constructs a lattice in coordinate space from several unit cells. The discrete scattered electric field components, exs, eys, and ezs, lie on the edges of each unit cell. For example, exs lies on the midpoint of edges in the x direction. The discrete magnetic field components, hxs, hys, and hzs, occur on the centers of faces of each unit cell. For example, hxs lies on the center of cell faces perpendicular to the x direction. Refer to Figure 3.5 of a typical unit cell for the location of the discrete field components. To every unit cell, GFDTD assigns a permittivity, a permeability, and a conductivity. Refer to Figure 3.6 of a lattice in coordinate space for the placement of a typical unit cell. Refer to [3-1], pp.302 to 307, for the Yee update method.

The finite difference form of Maxwell's curl equations determines the spatial and temporal update of the discrete field components on the lattice. At time, n, the magnetic fields, on cell faces and at half a time step back, update neighboring electric field components on cell edges. For example, GFDTD uses hys and hzs components to update exs components. GFDTD, then, increments time by half the time step, dt. The new electric field components, on cell edges and at the new time, n+(dt/2), update neighboring magnetic field components on cell centers. GFDTD, then, increments time again by half of dt. This process continues until the discrete field components have constant maximum amplitudes and constant phase relative to a fixed iteration number.

2. Parallel Finite Difference Time Domain Code

The Parallel Finite Difference Time Domain (PFDTD) code has features different from GFDTD.

PFDTD constructs a global lattice in coordinate space. This global lattice is identical to the lattice constructed by GFDTD. PFDTD divides this global lattice into blocks of nearly equal dimensions. PFDTD assigns neighboring blocks of the global lattice to nodes connected by a communication channel. This decomposition scheme assures that each node can perform its discrete field updates with resident information and information communicated by neighboring nodes.

Figure 3.7 illustrates a decomposition of the 7 x 7 x 7 unit cell lattice of Figure 3.6. In this example, PFDTD uses four nodes of the hypercube. PFDTD assigns a 7 x 4 x 4 cell block to node 0, a 7 x 3 x 4 cell block to node 1, a 7 x 4 x 3 cell block to node 2, and a 7 x 3 x 3 cell block to node 3. Figure 3.7 shows a cut in a plane perpendicular to the x axis. In this example, there are seven such planes along the x direction.

Maxwell's Equation for Scattered E Field

$$\varepsilon\, \partial_t E_x^s = -(\varepsilon - \varepsilon_o)\partial_t E_x^i - \sigma(E_x^i + E_x^s) + (\partial_y H_z^s - \partial_z H_y^s)$$

Finite Difference Form

$$E_x^{s,n}(i,j,k) - E_x^{s,n-1}(i,j,k) = (\Delta t/\varepsilon)\ \{ -(\varepsilon - \varepsilon_o)\partial_t E_x^{i,n}(i,j,k)$$
$$-\sigma E_x^{s,n-1}(i,j,k) + (H_z^{s,n-1/2}(i,j+1/2,k) - H_z^{s,n-1/2}(i,j-1/2,k)) / \Delta y$$
$$-\sigma E_x^{i,n}(i,j,k) -(H_y^{s,n-1/2}(i,j,k+1/2) - H_y^{s,n-1/2}(i,j,k-1/2)) / \Delta z \}$$

s : scattered field
i : incident field
n : current time step
i,j,k : Cartesian indices of lattice
Δt : time increment
Δy : cell length in the y direction

Figure 3.3 Maxwell's equation in finite difference form for the x component of the scattered electric field. The equations for the y and z components follow by cyclically permuting the x, y, and z subscripts and i, j, and k array indices.

Maxwell's Equation for Scattered H Field

$$\partial_t \mu H_x^s = -(\mu-\mu_o)\partial_t H_x^i - (\partial_y E_z^s - \partial_z E_y^s)$$

Finite Difference Form

$$H_x^{s,n}(i,j+1/2,k+1/2) - H_x^{s,n-1}(i,j+1/2,k+1/2) =$$
$$(\Delta t/\mu)\ \{\ -(\mu-\mu_o)\partial_t H_x^{i,n}(i,j+1/2,k+1/2)$$
$$-(\ E_z^{s,n-1/2}(i,j+1,k) - E_z^{s,n-1/2}(i,j,k)\ )\ /\ \Delta y$$
$$+(\ E_y^{s,n-1/2}(i,j,k+1) - E_y^{s,n-1/2}(i,j,k)\ )\ /\ \Delta z\ \}$$

s : scattered field
i : incident field
n : current time step
i,j,k : Cartesian indices of lattice
Δt : time increment
Δy :cell length in the y direction

Figure 3.4 **Maxwell's equation in finite difference form for the x component of the scattered magnetic field. The equations for the y and z components follow by cyclically permuting the x, y, and z subscripts and i, j, and k array indices.**

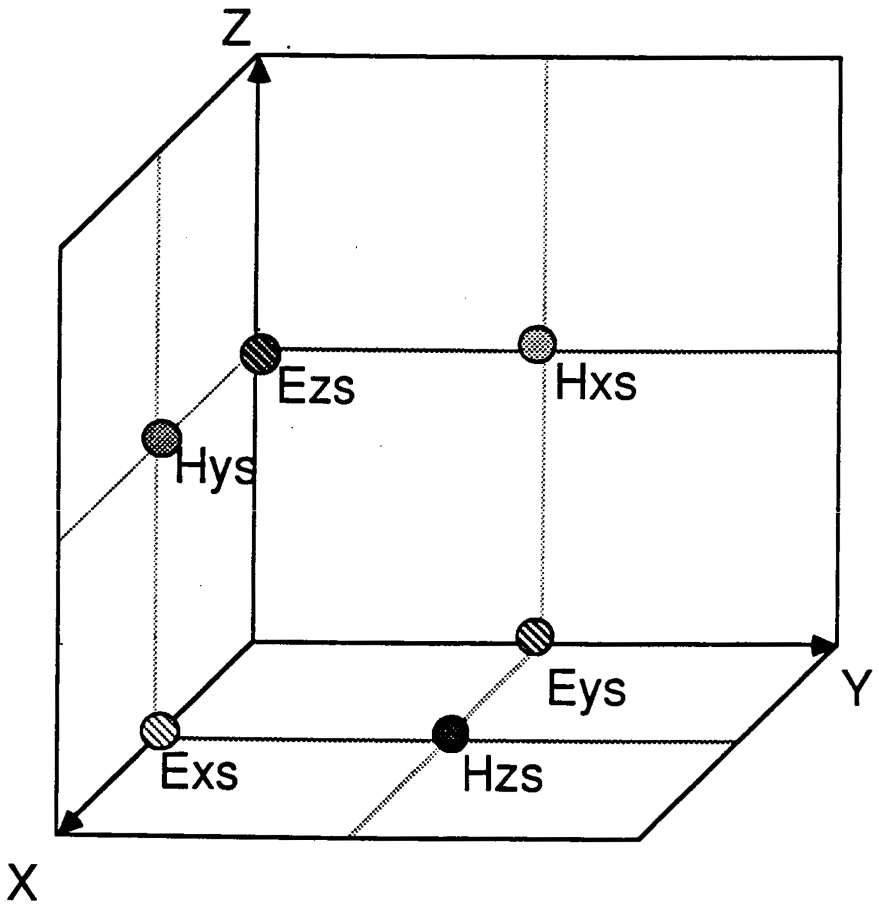


Figure 3.5 The location of electric and magnetic field components in a typical unit cell.

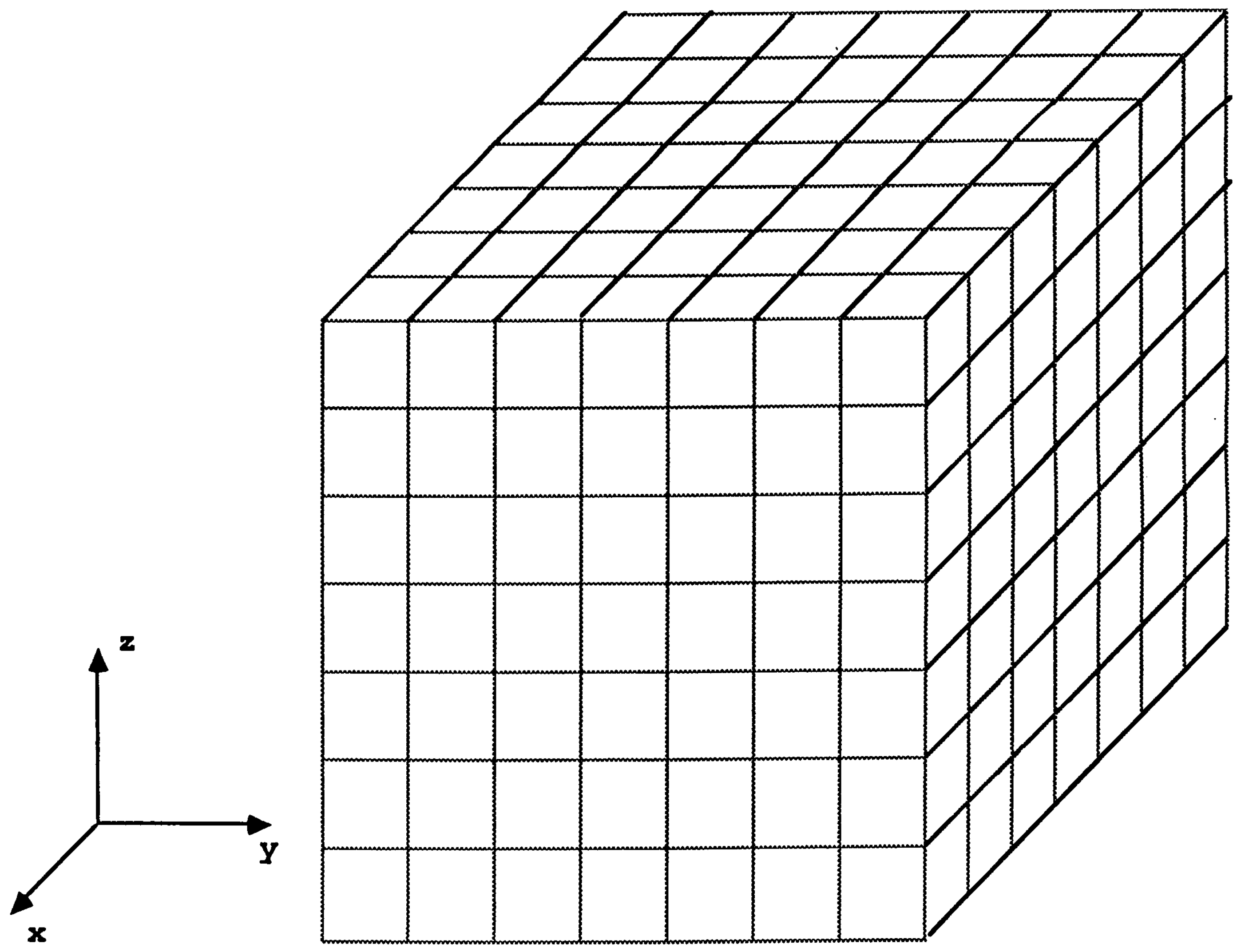


Figure 3.6 Adjacent unit cells in a global lattice.

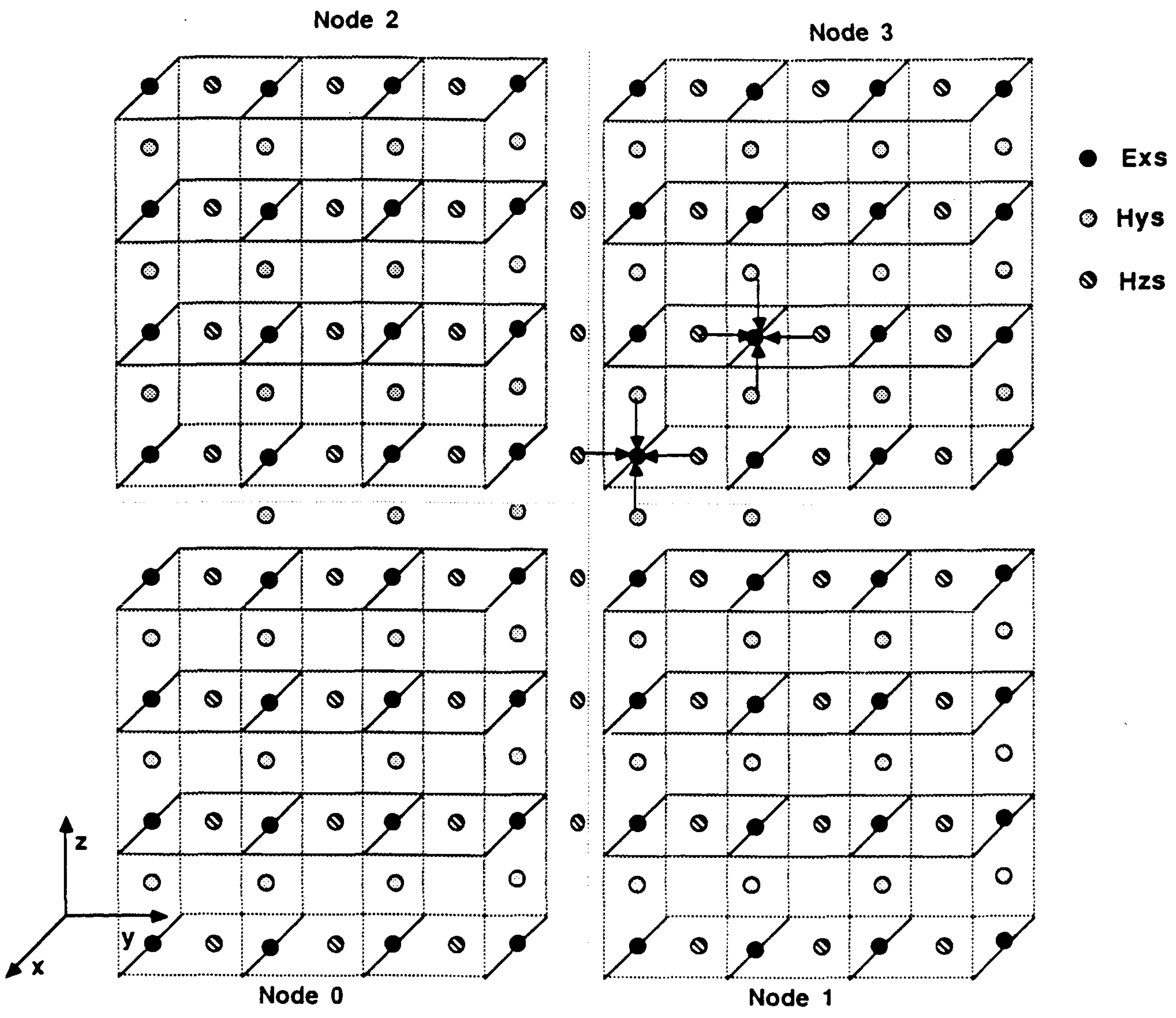


Figure 3.7 The decomposition of the global lattice among four nodes of the hypercube: magnetic field. Black arrows indicate the magnetic field components used to update x components of the electric field within a node and on the boundary between nodes.

With eight active nodes, PFDTD can also divide the global lattice in the x direction. The program will assign four cells in the x direction to the back four nodes and three cells to the front four nodes. In the x direction, the front four nodes have Cartesian coordinates greater than the back four nodes. The decomposition in the y direction for eys and in the z direction for ezs is identical to the decomposition described in the previous paragraph.

3. Update of the Electric and Magnetic Field Components

In the following paragraphs, we will explain the method to update the electric and magnetic field components within the volume of the global lattice. The x component of the electric field and the x component of the magnetic field will serve as examples. We cyclically permute the x, y, and z indices to generalize the discussion to the other four field components.

Figure 3.7 illustrates the update of the x component of the electric field, exs. To update the discrete values of exs on the lattice, PFDTD requires the values of exs at the previous time step, the values of neighboring hys and hzs discrete fields at half a time step back, and the value of the x component of the incident electric field at the current time step. For dielectric materials, PFDTD also requires the first time derivative of this incident field at the current time. For an exs component in the middle of a block assigned to a particular node, the update is the same as the sequential case. For an exs component on a boundary between nodes, neighboring nodes must communicate before the update. The black arrows pointing to a central exs component illustrate the two types of updates that can occur within node 3. Figure 3.3 shows the governing difference equation.

Figure 3.8 illustrates the update of the x component of the magnetic field, hxs. To update the discrete values of hxs on the lattice, PFDTD requires the value of hxs at the previous time step and the values of neighboring eys and ezs discrete fields at half a time step back. For dielectric materials, PFDTD also requires the first time derivative of the incident magnetic field at the current time. For an hxs component in the middle of a block assigned to a particular node, the update is same as the sequential case. For an hxs component on a boundary between nodes, neighboring nodes must communicate before the update. The black arrows pointing to a central hxs component illustrate the two types of updates that can occur within node 0. Figure 3.4 shows the governing difference equation.

4. Second Order Correct Radiation Boundary Conditions

Second order correct radiation boundary conditions work well with the spatial decomposition of the global lattice. A node, responsible for field points on the truncation planes, the planes marking the boundary of the computation lattice, has enough resident and communicated field information to update truncation plane points using second order correct radiation conditions.

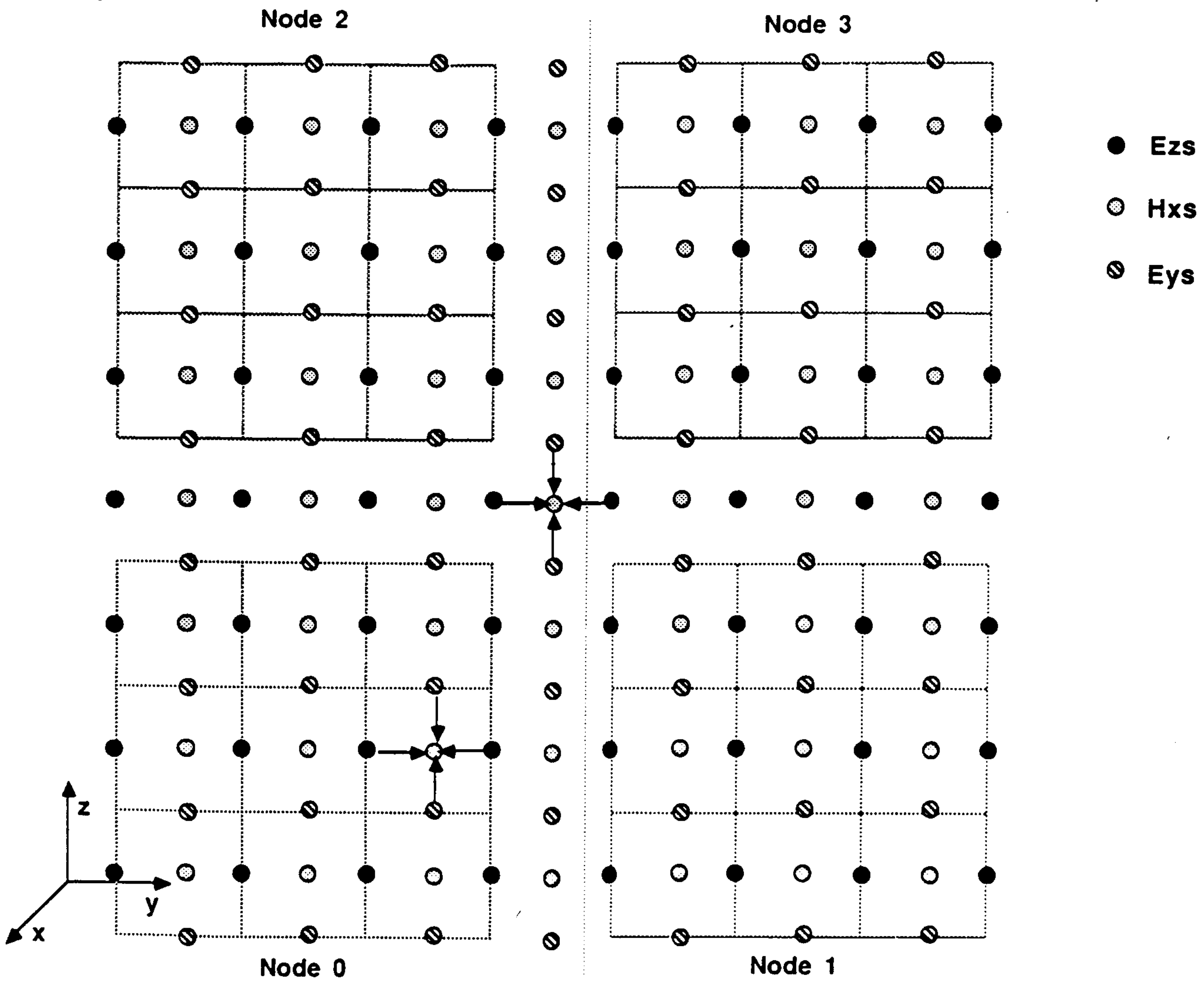


Figure 3.8 The decomposition of the global lattice among four nodes of the hypercube: electric field. Black arrows indicate the electric field components used to update x components of the magnetic field within a node and on the boundary between nodes.

Figure 3.9 illustrates the update of an x component of the electric field on a truncation plane perpendicular to the y axis and with unit normal vector in the positive y direction. We label this point with the number 1. We cut a section perpendicular to the x axis to illustrate the update of point 1. To update point 1 at time n, PFDTD needs the value of the field at points 1 and 2 at time 2*dt back and time dt back. The program needs the value of the field at points 3 to 10 at time dt back. If any of the points 3 to 10 reside in neighboring nodes, the node containing point 1 must communicate with its neighbors before the update occurs.

Figure 3.10 gives the relevant equation used in the update of exs points on this truncation plane. We remove the superscript, s, indicating scattered fields, for convenience. There exist similar equations for exs on truncation planes perpendicular to the z axis, for eys on truncation planes perpendicular to the x and z axes and for ezs on truncation planes perpendicular to the x and y axes.

5. Radar Cross Section

The spatial decomposition of the global lattice also works well for calculating the normalized radar cross section. Several nodes in the hypercube contribute to the integration that yields the electric and magnetic vector potentials. PFDTD combines the local contributions to these potentials to calculate E theta; E phi; the radar cross section, sigma; and the normalized radar cross section RCS. Figure 3.11 illustrates the relevant equations.

The variables in Figure 3.11 equations are the following: sigma is the radar cross section, r is the distance of the observation point, r prime is the distance of the source point, a is the radius of the sphere, k naught is the wave number of the incident field, eta naught equals 376.73 ohms, theta and phi are the angles of the observation point, and E and H are the complex fields obtained from magnitude and phase evaluations on the truncation planes.

The radar cross section calculation depends on the magnitude and phase of discrete field components on an integration surface. PFDTD uses a cube surrounding the scattering object as the integration surface. Portions of the cubic integration surface reside in different nodes. The program obtains the magnitude of field components on this integration surface by recording the peak and valley of the steady state sinusoidal wave forms. When a peak occurs, the program also records the time step. From this time step information, the time step increment dt, the wave number of the incident field, and the speed of light in vacuum, PFDTD calculates the phase relative to a reference time step.

With the magnitude and phase of these near field points, PFDTD calculates rcs. Each node performs the vector potential integration on its portion of the integration surface. Each node calculates a local contribution to the fields E theta and E phi. PFDTD sums these local contributions in the control processor to obtain the global total E theta and E phi. With these total fields, the program calculates the radar cross section, sigma, and the normalized radar cross section, rcs.

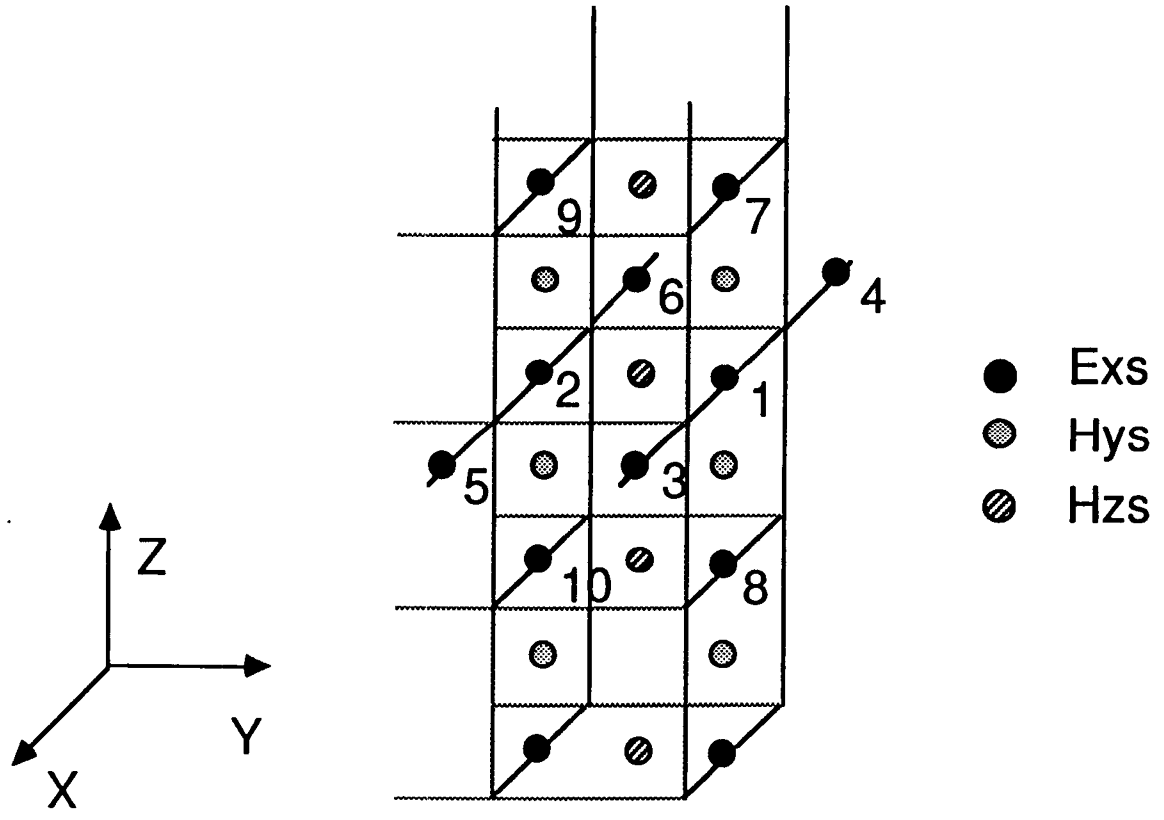


Figure 3.9 Field points required to update an x component of the electric field on a truncation plane perpendicular to the y axis with unit normal vector in the positive y direction. The update is done using second order correct radiation boundary conditions.

$$
\begin{aligned}
E_x^{n+1}(i,ny+1,k) &= E_x^{n-1}(i,ny,k) \\
&+ \frac{c\Delta t - \Delta y}{c\Delta t + \Delta y}\left\{ E_x^{n+1}(i,ny,k) + E_x^{n-1}(i,ny+1,k) \right\} \\
&+ \frac{2\Delta y}{c\Delta t + \Delta y}\left\{ E_x^{n}(i,ny,k) + E_x^{n}(i,ny+1,k) \right\} \\
&+ \frac{\Delta y\,(c\Delta t)^2}{2(\Delta x)^2\,(c\Delta t + \Delta y)}\Big\{ E_x^{n}(i+1,ny+1,k) - 2E_x^{n}(i,ny+1,k) \\
&\qquad + E_x^{n}(i-1,ny+1,k) + E_x^{n}(i+1,ny,k) - 2E_x^{n}(i,ny,k) + E_x^{n}(i-1,ny,k) \Big\} \\
&+ \frac{\Delta y\,(c\Delta t)^2}{2(\Delta z)^2\,(c\Delta t + \Delta y)}\Big\{ E_x^{n}(i,ny+1,k+1) - 2E_x^{n}(i,ny+1,k) \\
&\qquad + E_x^{n}(i,ny+1,k-1) + E_x^{n}(i,ny,k+1) - 2E_x^{n}(i,ny,k) + E_x^{n}(i,ny,k-1) \Big\}
\end{aligned}
$$

n : current time step
i,j,k : Cartesian indices of lattice
Δt : time increment
Δy : cell length in the y direction
ny : index of last cell in the y direction

Figure 3.10 Second order correct radiation boundary equation to update x components of the electric field on a truncation plane perpendicular to the y axis and with unit normal vector in the positive y direction. Similar equations exist for exs on truncation planes perpendicular to the x and z axes, and for ezs on planes perpendicular to the x and y axes.

$$\sigma = 4\pi r^2 \frac{E_\theta^* E_\theta + E_\phi^* E_\phi}{E^{i*} E^i} \quad r \to \infty \qquad \mathrm{rcs} = \frac{\sigma}{4\pi a^2}$$

$$E_\theta = -ik_o \eta_o \left(A_x \cos\theta \cos\varphi + A_y \cos\theta \sin\varphi - A_z \sin\theta \right.$$
$$\left. + \eta_o^{-1} \left(-F_x \sin\varphi + F_y \cos\varphi \right) \right)$$

$$E_\phi = -ik_o \eta_o \left(-\eta_o^{-1} \left(F_x \cos\theta \cos\varphi + F_y \cos\theta \sin\varphi - F_z \sin\theta \right) \right.$$
$$\left. - A_x \sin\varphi + A_y \cos\varphi \right)$$

$$\vec{A} = \frac{e^{-ik_o r}}{4\pi r} \iint_s \left(\vec{n} \times \vec{H}^s \right) e^{ik_o r' \cos\xi} \, ds$$

$$\vec{F} = \frac{e^{-ik_o r}}{4\pi r} \iint_s \left(-\vec{n} \times \vec{E}^s \right) e^{ik_o r' \cos\xi} \, ds$$

$$r' \cos\xi = (x' \cos\varphi + y' \sin\varphi) \sin\theta + z' \cos\theta$$

Figure 3.11 The radar cross section, sigma, and the normalized radar cross section are in terms of E theta and E phi. E theta and E phi are in terms of the vector potentials, A and F. The integrations are over a cubic surface surrounding the scattering object. The complex H and E fields are the magnitude and phase of the real fields on the integration surface.

C. RESULTS FOR A PERFECTLY CONDUCTING CUBE

To test the validity of the results from PFDTD, we compared the program's reported current on the surface of a perfectly conducting cube with the results reported on page 163 of [3-3].

We illustrate the scattering object in Figure 3.12. The scattering object is a perfectly conducting cube in vacuum. The wave number times the length of a cube side, ks, equals 2. The cube is one meter in each direction. A plane wave is incident on the cube's front face. The direction of propagation of this plane wave is the z direction. The electric field is polarized in the y direction with magnitude 376.8 V/m. The magnetic field is polarized in the -x direction with magnitude 1.0 A/m. The wave number of this incident field is 2 radian/m. The frequency is 95.43 MHz. The wavelength is 3.141593 m. The plane wave hits the cube at time t=0.

We choose a path along the bottom of the cube to evaluate the x component of the total magnetic field. Knowing this magnetic field component at the nine selected test points, we can find the magnitude and phase of the tangential current along the path at these points. The current on the surface of a perfect conductor is $\bar{J} = \bar{n} \times \bar{H}$, where $\bar{n}$ is the unit normal vector to the surface of the scatterer.

Because the memory limitation of each node of the hypercube affects the size of our test runs, we mention the array limitations of the version of PFDTD which we use for these runs. For code development reasons, the program uses double precision arrays. The arrays are double precision because we converted the original code, Generalized Finite Difference Time Domain (GFDTD), from Lawrence Livermore National Laboratory, which contained double precision arrays. The 4 Mbytes of memory on each node of the hypercube limit the code to allocate at most 37 x 37 x 37 unit cell blocks in each processor. With 32 nodes, 74 x 74 x 74 unit cells is the maximum size of a square lattice. With 0.05 x 0.05 x 0.05 m. unit cells, the maximum size for a square lattice is 3.7 x 3.7 x 3.7 m.

The current version of PFDTD uses single precision arrays with the added capability to calculate radar cross section. With 32 nodes, the maximum size of a square lattice for this new version is 80 x 80 x 80 unit cells. With 32 nodes, 160 x 160 x 80 unit cells is the maximum size of a non-cubic lattice.

The parameters for the cube scatterer and for the incident field are identical to those values used by Taflove. We are free to specify the lattice and unit cell sizes. For example, in the run that generated the plots in Figure 3.13, the lattice truncation planes are 0.5 m from the surface of the scatterer. The unit cell has length 0.05 x 0.05 x 0.05 m. The time step, dt, is related to this length by the Courant stability condition. Dt equals 0.09629 ns.

In Figure 3.13, the results from PFDTD compare poorly with Taflove's result. We graph Taflove's result as solid lines in the magnitude and phase plots. The dashed curves are results from PFDTD. For this run, PFDTD updates the points on the truncation planes using first order correct radiation boundary conditions. Because the truncation planes are 0.5 m from the

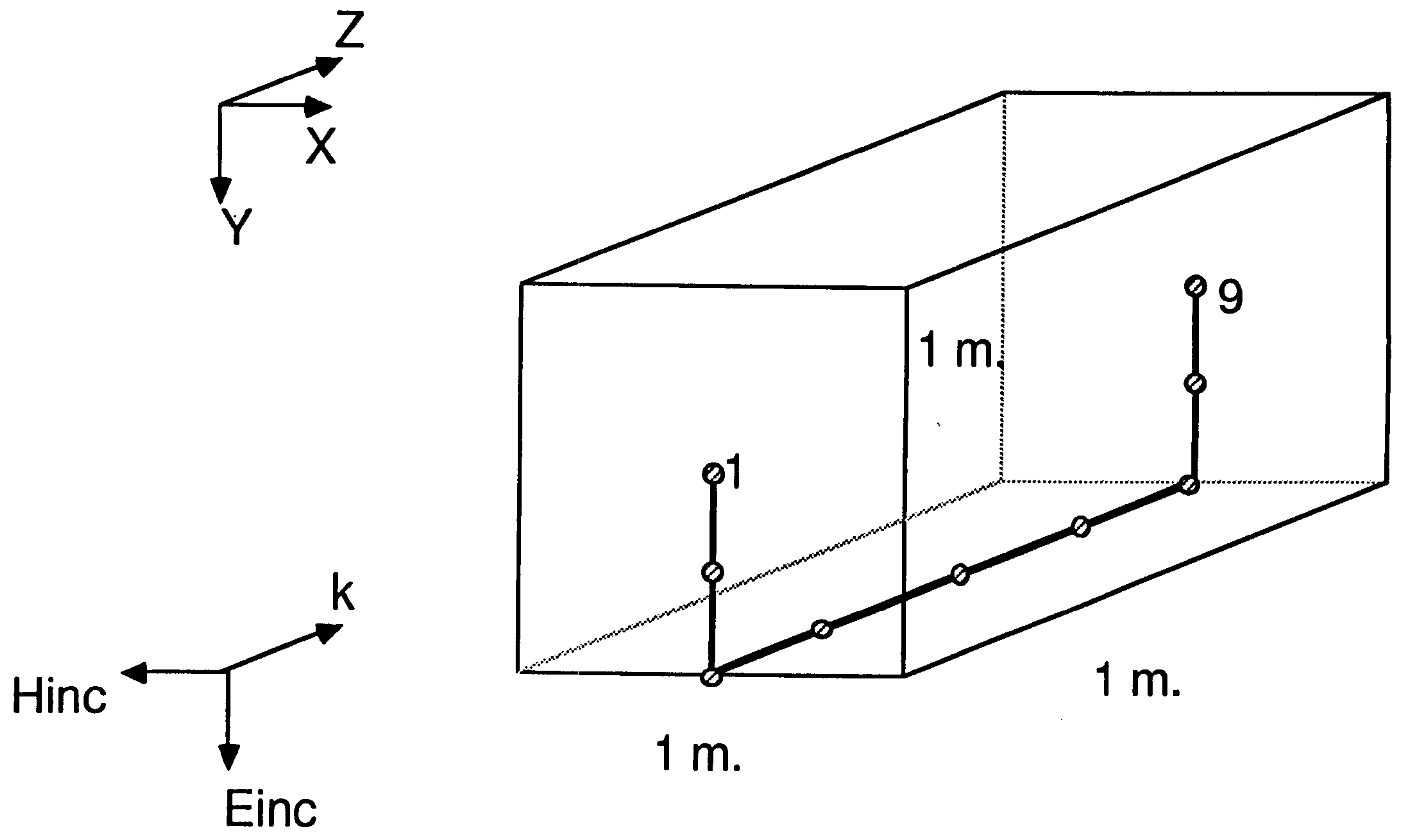


Figure 3.12 The scattering object is a perfectly conducting cube in vacuum. The wave number times the length of a cube side is 2. A plane wave is incident on the cube's front face with direction of propagation in the z direction, electric field polarized in the y direction with magnitude 376.8 V/m, and magnetic field polarized in the -x direction with magnitude 1.0 A/m. The wave number for this incident field is 2 radian/m.

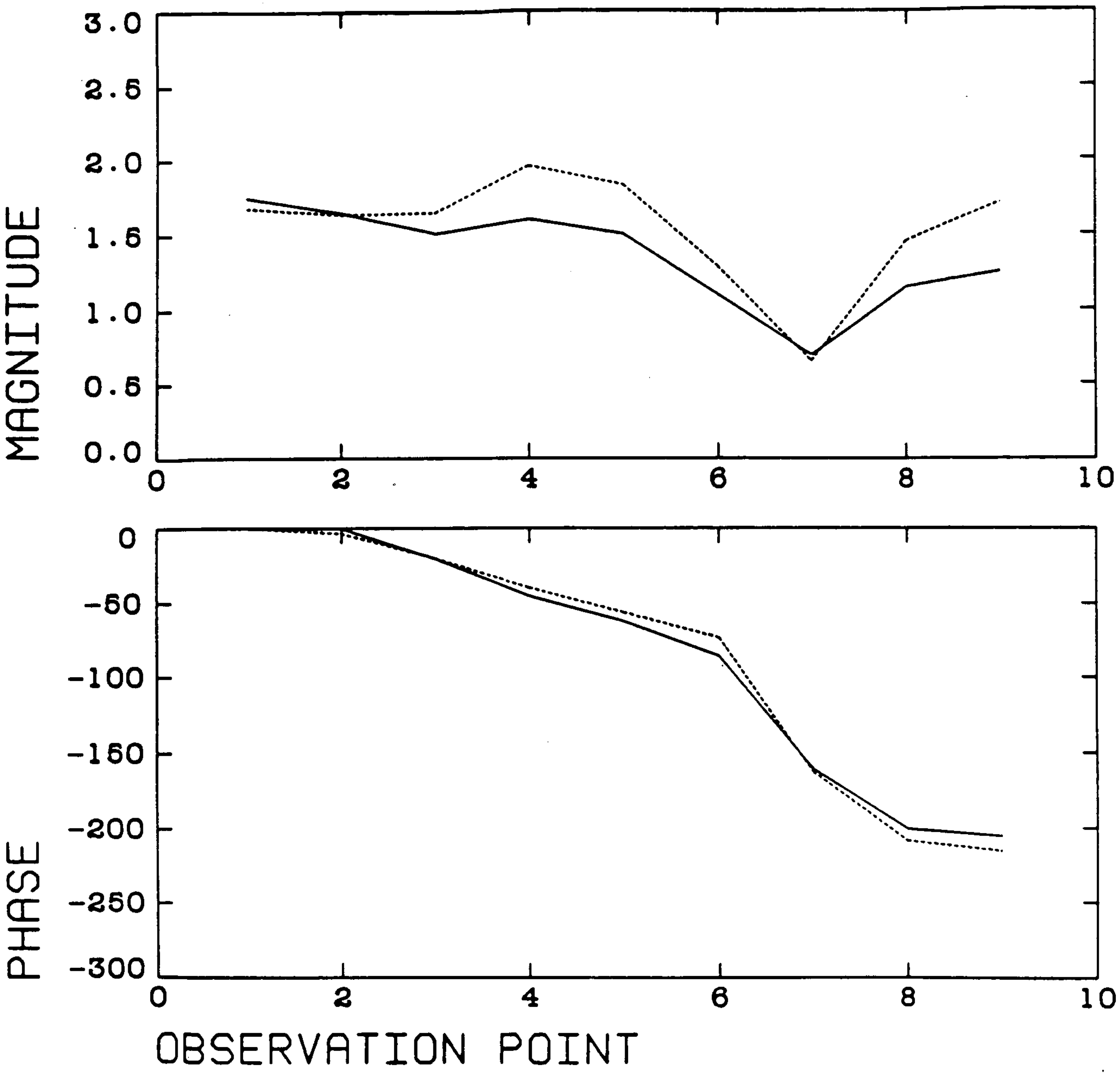


Figure 3.13 The magnitude and phase from Taflove's results of scattering from a perfectly conducting cube and from a particular run using PFDTD: first order correct radiation boundary conditions. The solid lines show Taflove's results. The dashed lines show PFDTD results. We evaluate the current on the cube at nine test points which we indicate in Figure 3.12. For this run, PFDTD updates points on the truncation planes using first order correct radiation boundary conditions. The lattice truncation planes are 0.5 m from the surface of the scatterer. The unit cell has length 0.05 m. The time step is 0.09629 ns.

scattering surface, the magnitude and phase of the current from PFDTD does not compare well with the Taflove result. When the distance of the truncation planes increases to 1.4 m from the surface of the scatterer, the PFDTD curves for magnitude and phase do not differ from the Taflove result by more than 10% in the non-shadow region of the scatterer.

The results graphed in Figure 3.14 show better agreement. The parameters for the scatterer and the incident field for Figure 3.14 are identical to the parameters for Figure 3.13. However, PFDTD uses second order correct radiation boundary conditions. With the truncation planes 0.5 m from the surface of the 1.0 m cube, the PFDTD curves for magnitude and phase do not differ from the Taflove result by more than 10%. Second order correct radiation boundary conditions significantly decreased the difference between the Taflove curves and the PFDTD curves.

To make sure that second order boundary conditions produce sensible results, we increase the size of the lattice. The larger lattice size should not significantly alter the curves shown in Figure 3.14. The parameters for the scatterer and the incident field are identical to the two previous runs. However, the truncation planes moved to 1.35 m. from the surface of the scatterer. Figure 3.15 shows the results of this run. Because the PFDTD solid curves have not changed significantly from the curves on Figure 3.14, we conclude the following for the cube scatterer: second order radiation boundary conditions significantly reduce reflections from truncation planes within a distance of 0.5 times the length of the scattering object. The following test runs also support this conclusion.

Figure 3.16 is the result of our attempts to match the Taflove curves more closely and to test the action of the boundary conditions. The unit cell size is 0.03333 x 0.03333 x 0.03333 m. The time step equals 0.06419 ns. The lattice size is 2.0 x 2.0 x 2.0 m. Except for the shadow region, behind the scattering cube, the PFDTD curves for amplitude and phase do not differ from the Taflove result by more than 6%. We also ran the case with a unit cell size of 0.027778 x 0.027778 x 0.027778 m. The time step equals 0.05346 ns. The lattice size is now 2.06 x 2.06 x 2.06 m. The magnitude and phase curves for this run are similar to curves in Figure 3.16. In the non-shadow regions, the PFDTD and Taflove curves do not differ by more than 6%.

D. PERFORMANCE OF THE HYPERCUBE

To analyze the performance of PFDTD, we must identify the computing intensive parts of the code. PFDTD contains input/output subroutines and initialization and setup subroutines. Depending on the needs of the user, these subroutines typically last no more than 90 seconds. However, the time step iteration loop, in which the program performs field updates, radiation boundary condition updates, and internode communication for several iterations, can consume 90 minutes. Therefore, we concentrate on the execution times for the various parts of this iteration loop.

There exist two methods to measure the efficiency of the program, PFDTD. The first method of measurement fixes the problem size and increases the number of active nodes. If the program were 100% efficient and if the number

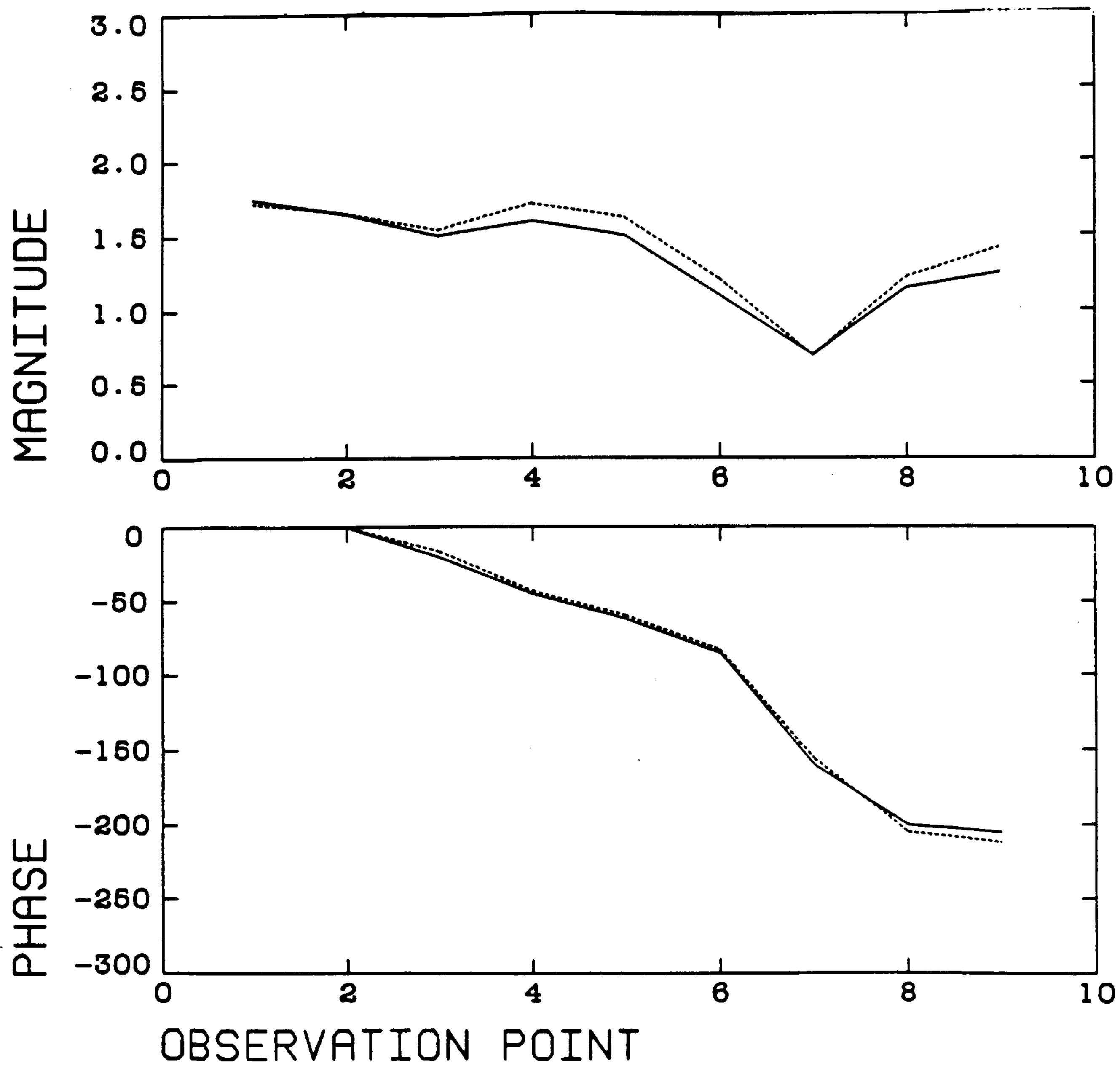


Figure 3.14. The magnitude and phase from Taflove's results of scattering from a perfectly conducting cube and from a particular run using PFDTD: second order correct radiation boundary conditions. The solid lines show Taflove's results. The dashed lines show PFDTD results. We evaluate the current on the cube at nine test points which we indicate in Figure 3.12. For this run, PFDTD updates points on the truncation planes using second order correct radiation boundary conditions. The lattice truncation planes are 0.5 m from the surface of the scatterer. The unit cell has length 0.05 m. The time step is 0.09629 ns.

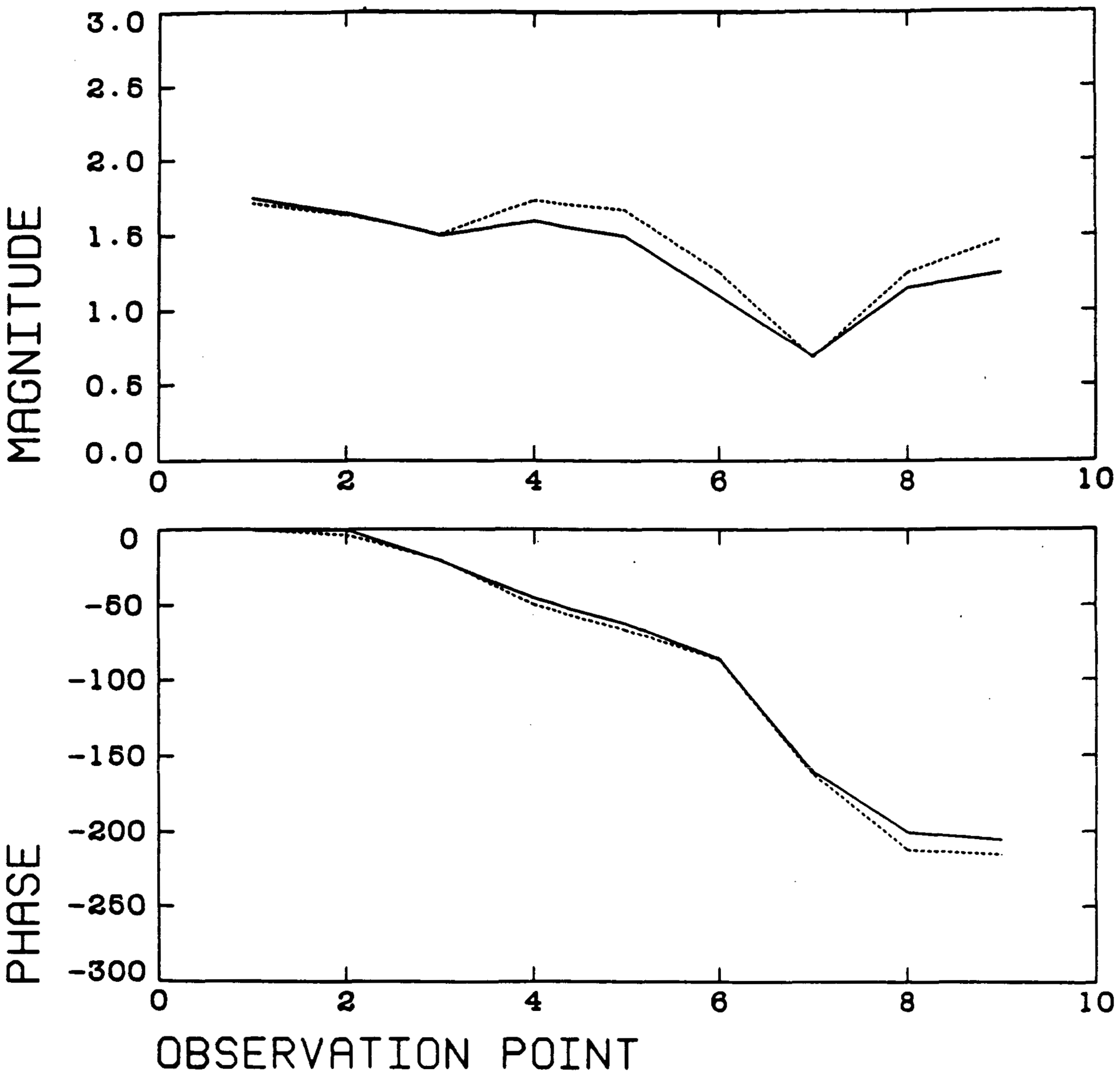


Figure 3.15 The magnitude and phase from Taflove's results of scattering from a perfectly conducting cube and from a particular run using PFDTD: the lattice truncation plane moved to 1.35 m from the surface of the scatterer. The solid lines show Taflove's results. The dashed lines show PFDTD results. We evaluate the current on the cube at nine test points which we indicate in Figure 3.12. For this run, PFDTD updates points on the truncation planes using second order correct radiation boundary conditions. The lattice truncation planes are 1.35 m from the surface of the scatterer. The unit cell has length 0.05 m. The time step is 0.09629 ns.

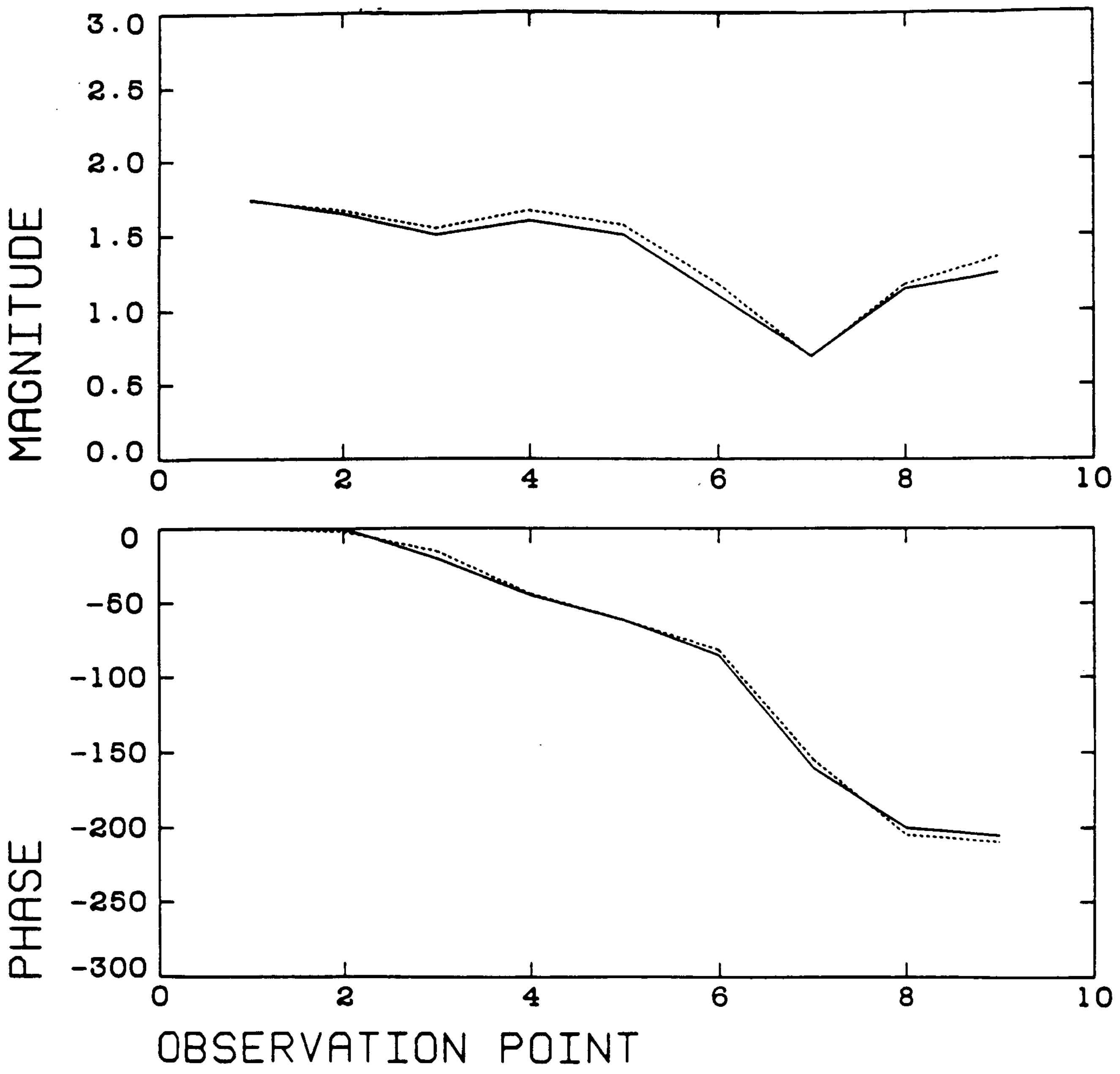


Figure 3.16 The magnitude and phase from Taflove's results of scattering from a perfectly conducting cube and from a particular run using PFDTD: unit cell length 0.03333 m. The solid lines show Taflove's results. The dashed lines show PFDTD results. We evaluate the current on the cube at nine test points which we indicate in Figure 3.12. For this run, PFDTD updates points on the truncation planes using second order correct radiation boundary conditions. The lattice truncation planes are 0.5 m from the surface of the scatterer. The unit cell has length 0.03333 m. The time step is 0.06419 ns.

of active nodes increases by a factor N, the execution time of the code should drop by 1/N. However, the addition of more nodes also increases the amount of internode communication. If this communication time is a significant fraction of the calculation time, poor efficiency will result. To assure that the ratio of the communication time to the calculation time is small, the global lattice must contain a large number of unit cells. Because the large lattice size demands the update of many discrete field components, the calculation time should dominate the communication time. However, a large lattice size may prevent PFDTD from running on small node ensembles of 1, 2, 4, or 8 processors.

The second method of efficiency measurement fixes the problem size in each node while increasing the number of active nodes. If the code were 100% efficient and if the number of active nodes increases by a factor N, the execution time of the code should remain constant because the total computational load also increased by a factor N. However, the time may not remain constant because of the added internode communication. Note that, with each increase in the number of active processors, PFDTD solves a different problem because the global lattice size increases.

To assure that each node does about the same amount of work for the purposes of timing, we instruct PFDTD to free the entire global lattice of scattering material. This instruction is necessary to easily scale the problem. This condition places limitations only on the field update calculations. Instead of updating the fields according to perfectly conducting or dielectric material difference equations, PFDTD updates the fields according to vacuum difference equations.

We concentrate on the various components in the iteration loop and utilize both methods of efficiency measurement. In each time step iteration, the program updates the electric and magnetic field components, communicates electric and magnetic field components on boundaries between nodes, and performs second order radiation boundary updates. Depending on the users needs, the program, at each iteration, may perform near field interpolation or evaluate magnitude and phase on integration planes. The execution times of these two activities have magnitudes similar to the time for internode communication. We do not include these execution times in the following plots because PFDTD does not perform these calculations at every time step and because these execution times are small compared to the times for field updates and radiation boundary updates.

Table 3.1 shows three sets of timing runs using the first method of efficiency measurement. For the first set of times, we fix the global lattice size at 64000 cells. We record the maximum reported execution time per iteration for each component of the time step loop, which we described in the previous paragraph, the total time per iteration, the speedup, and the efficiency. We record times using 1, 2, 4, 8, 16, and 32 active nodes. For this first set, speedup equals run time using one node divided by run time using multiple nodes. For this set, efficiency equals speedup divided by the number of active nodes.

We comment on this first set of timing runs. As the number of active nodes increases by a factor of two, the time per iteration to update field components within the volume of the computation lattice decreases by a factor

Table 3.1 Three sets of timing runs using the same number of cells in the computation lattice for given set. In the first set, we use 64000 unit cells. In the second set, we use 512000 unit cells. In the last set, we use 1024000 unit cells. The columns indicate the time per iteration in seconds for field updates within the computation lattice, for boundary updates on the lattice truncation planes, and for internode communication; the total time per iteration; the speedup and efficiency, as defined in the text; and the number of active nodes.

64000 unit cells

Update Fields	Boundary Conditions	Communication	Total Time/Iteration	Speedup	Efficiency	Number of Active Nodes
37.300	5.750	0.001	43.125	1.00	1.00	1
19.313	3.517	0.395	22.464	1.92	0.96	2
9.582	1.812	0.235	11.248	3.83	0.96	4
4.896	1.031	0.185	5.783	7.46	0.93	8
2.491	0.668	0.153	3.088	13.97	0.87	16
1.248	0.460	0.099	1.699	25.38	0.79	32

512000 unit cells

Update Fields	Boundary Conditions	Communication	Total Time/Iteration	Speedup	Efficiency	Number of Active Nodes
39.022	4.094	1.069	42.340	1.00	1.00	8
19.800	2.701	0.775	21.912	1.93	0.97	16
9.907	1.807	0.495	11.417	3.71	0.93	32

1024000 unit cells

Update Fields	Boundary Conditions	Communication	Total Time/Iteration	Speedup	Efficiency	Number of Active Nodes
39.479	4.052	1.330	42.489	1.00	1.00	16
19.747	2.903	0.920	22.094	0.92	0.96	32

of two. The times in the next two columns include the time per iteration to update fields on the truncation planes and to perform internode communication. These times increase with the number of active nodes. For 16 and 32 nodes, the number of cells per node are 4000 and 2000. In the 16 and 32 node configurations, the ratio of communication time over calculation time is higher compared to the same ratio using 2 and 4 nodes. This higher ratio results in decreased efficiency.

In the next set of timing runs in Table 3.1, we fix the global lattice size at 512000 cells. Because of memory limitations in each node, we record time using only 8, 16, and 32 active nodes. For this second set, we define speedup as the run time using eight nodes divided by the run time using multiple nodes. Efficiency equals speedup multiplied by eight and divided by the number of active nodes. For 16 and 32 nodes, the numbers of cells per node are 256000 and 128000. This high density of cells per node results in high efficiency compared to the 8 node configuration.

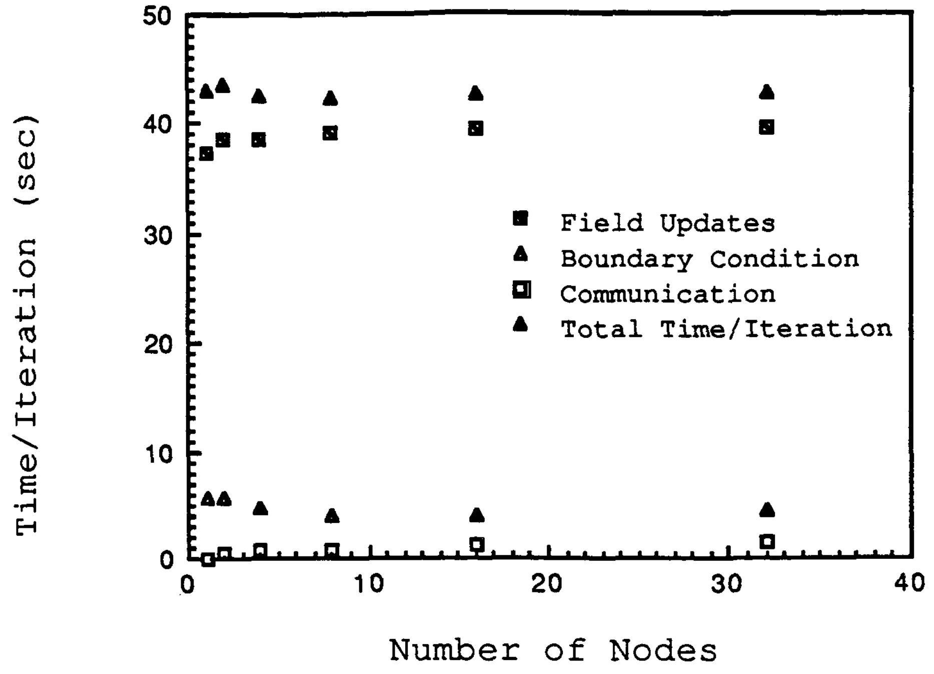
In the last set of timing runs in table 3.1, we fix the global lattice size at 1024000 cells. Because of memory limitations in each node, we record time using only 16 and 32 active nodes. For this third set, we define speedup as the run time using sixteen nodes divided by the run time using multiple nodes. Efficiency equals speedup multiplied by sixteen and divided by the number of active nodes. For 32 nodes, the number of cells per node is 512000. Again, the efficiency is high compared to the 16 node configuration.

Figure 3.17 shows two sets of timing runs using the second method for efficiency measurement. The horizontal axes indicate the number of active processors: 1, 2, 4, 8, 16, or 32. The vertical axes show the execution time in seconds. Hollow squares indicate the maximum reported internode communication time per iteration. Because nodes responsible for the boundaries of the global lattice may communicate less, nodes report distinct internode communication time. Hollow triangles indicate the maximum reported time per iteration to perform second order correct boundary updates. Squares with diagonal lines indicate the maximum reported time per iteration to perform electric and magnetic field updates. Lastly, filled triangles indicate the total time per iteration. We used 64000 unit cells per node in the runs that produced the values plotted in the top graph. We used 27000 unit cells per node in the runs that produced the values in the bottom graph.

Figure 3.18 shows two additional sets of timing runs. We used 8000 unit cells per node in the runs that produced the values plotted in the top graph. We used 1000 unit cells per node in the runs that produced the values in the bottom graph.

The various components of the iteration loop predictably follow certain trends. The communication time per iteration increases with the number of processors. The communication time is also a small fraction of the total time per iteration and is a small fraction of the time per iteration to perform field updates.

The communication time should hit an upper limit for a given density of cells per node because a node can communicate in at most six directions and because the amount of communicated information in a given direction remains constant for a fixed density. The existence of this upper limit is possible



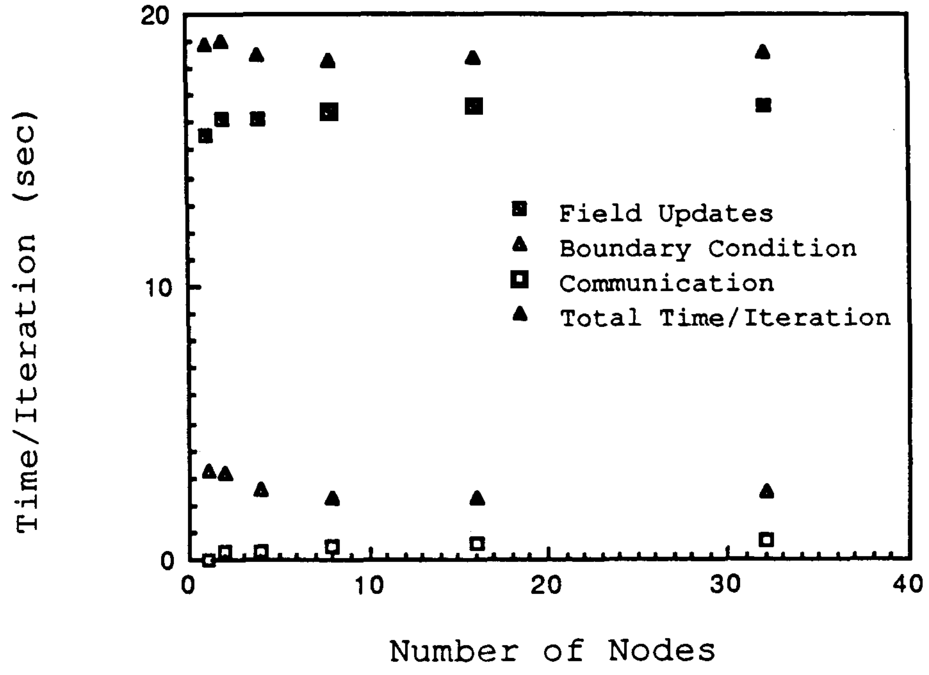
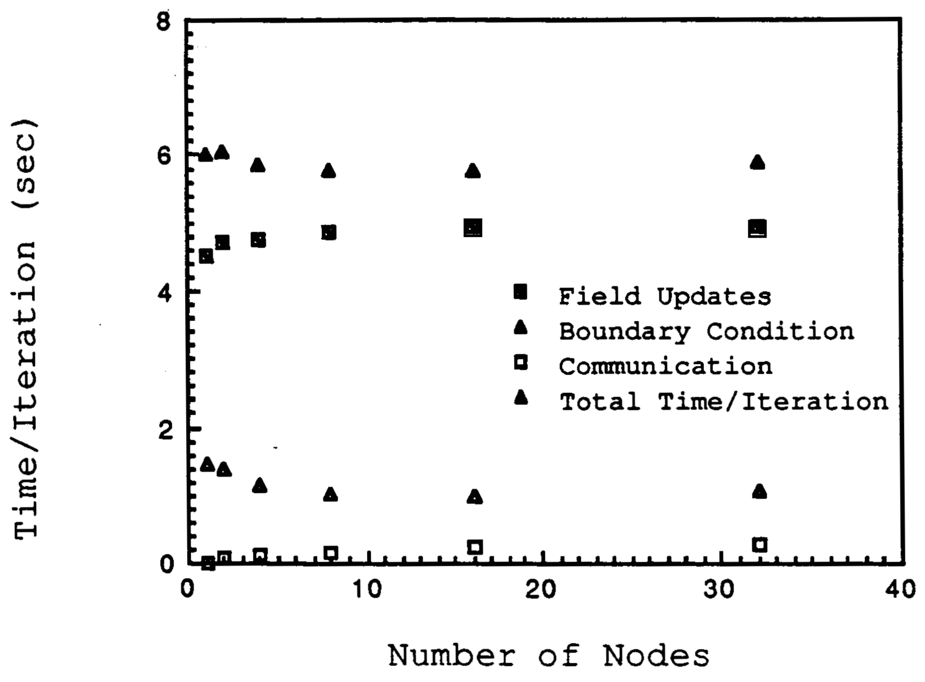


Figure 3.17 Two sets of timing runs for fixed number of cells per node for cases 64000 and 27000 unit cells per node. The horizontal axes indicate the number of active nodes. The vertical axes show the execution time in seconds. Symbols indicate the maximum reported time per iteration for field updates within the volume of computation lattice, for second order correct updates on the truncation planes, and for internode communication. The last symbol indicates the total time per iteration. The top graph plots points for 64000 unit cells per node. The bottom graph plots points for 27000 unit cells per node.



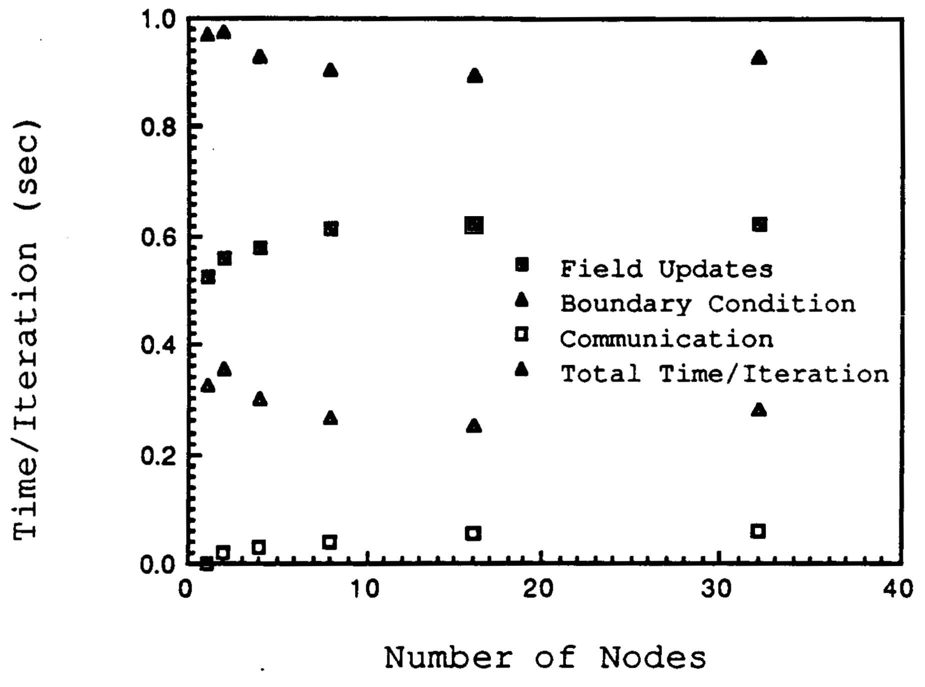


Figure 3.18 Two sets of timing runs for fixed number of cells per node for cases 8000 and 1000 unit cells per node. The horizontal axes indicate the number of active nodes. The vertical axes show the execution time in seconds. Symbols indicate the maximum reported time per iteration for field updates within the volume of computation lattice, for second order correct updates on the truncation planes, and for internode communication. The last symbol indicates the total time per iteration. The top graph plots points for 8000 unit cells per node. The bottom graph plots points for 1000 unit cells per node.

if nodes do not wait while other nodes in the configuration communicate. All nodes involved in a communication step should first write a packet of information, read the packet sent by its neighbor, and, then, continue to send and read additional packets. The current version of PFDTD does not have this communication scheme.

The second order boundary condition times from the above mentioned plots also follow predictable trends. As the number of nodes increases, the time per iteration to perform boundary condition updates decreases for node configurations of 2, 4, 8, and 16. For 32 active nodes, the time per iteration increases from the time reported for 16 nodes. Although the number of unit cells remains constant in each node as we increase the number of nodes, PFDTD still breaks up the truncation planes among the nodes on the periphery of the global lattice. This division is the reason for the decreased times in 2, 4, 8, and 16 nodes. Second order boundary condition updates also require internode communication. This communication is the reason for the increased time in 32 nodes.

The field update times follow predictable trends. There is a slight increase in the time per iteration for field updates as the number of nodes increases. As the number of nodes increases, a node performs less boundary condition updates. Because the number of cells remains constant, the node must now perform additional field updates within the volume of the computation lattice.

For a fixed density of cells per node, the four plots indicate that these competing times result in faster total execution time per iteration in 4, 8, 16 and 32 nodes compared to 1 and 2 nodes. By the definition of efficiency for this second method of measurement, we can make the statement that for 4, 8, 16, and 32 nodes PFDTD can perform with over 100% efficiency. With the above mentioned improvements in internode communication, this efficiency should remain high for arbitrarily large node configurations.

We also performed timing comparisons with a VAX 750 and a Counterpoint System XIX computer. The Counterpoint computer serves as the control processor. For approximately 64000 cells in the global lattice, the VAX 750 takes 47.47 seconds per iteration and the Counterpoint System XIX takes 89.81 seconds per iteration. The top plot in Figure 3.17 indicates that 1 node of the hypercube takes 43.14 seconds per iteration for the 64000 cell per node case.

We must qualify the above machine times. Because the VAX and Counterpoint are multi-user machines, the runs on these computers were done during hours of minimal usage. The code that runs on the VAX and Counterpoint is GFDTD. The code that runs on the hypercube is PFDTD. PFDTD contains all the enhancements, which include second order correct radiation boundary conditions. GFDTD still uses first order correct radiation boundary conditions. Because second order is computationally more demanding than first order, we expected the time per iteration on one node would surpass the time per iteration on the VAX and Counterpoint. However, the speed of the hypercube nodes actually allows a faster execution time.

REFERENCES

[3-1] K. S. Yee, "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media," IEEE Trans. Antennas Prop., Vol. AP-14, May 1966, pp. 302-307.

[3-2] A. Taflove and M. E. Brodwin, "Numerical Solution of Steady-State Electromagnetic Scattering Problems Using the Time-Dependent Maxwell's Equations," IEEE Trans. Microwave Theory Tech., Vol. MTT-23, August 1975, pp. 623-630.

[3-3] K. R. Umashankar and A. Taflove, "Analytical Models for Electromagnetic-Scattering," Part II: Finite-Difference Time-Domain Developments, Final Report on IITRI Project E06538, Electronics Department, IIT Research Institute, June 1984.

[3-4] K. S. Kunz, "Generalized Three-Dimensional Experimental Lightning Code (G3DXL) User's Manual," Kunz Associates, Inc., February 1986.

[3-5] A. Taflove and K. R. Umashankar, "A Hybrid Moment Method/Finite Difference Time Domain Approach to Electromagnetic Coupling and Aperture Penetration into Complex Geometries," IEEE Trans. Antennas Prop., Vol. AP-30, July 1982, pp. 617-627.

[3-6] G. Mur, "Absorbing Boundary Conditions for the Finite-Difference Approximation of the Time-Domain Electromagnetic-Field Equations," IEEE Trans. Electromag. Compat., Vol. EMC-23, November 1981, pp. 377-382.

[3-7] B. Engquist and A. Majda, "Absorbing Boundary Conditions for the Numerical Simulation of Waves," Math Comp., Vol. 31, July 1977, pp. 629-651.

SECTION IV

FREQUENCY DOMAIN CODE (NEC)

A. INTRODUCTION

The Numerical Electromagnetic Code (NEC-2) is used for the analysis of the electromagnetic response of antennas and other metallic structures. This code computes the induced current on structures modeled by small wires or surface patches. Other near-field or far-field quantities such as electric or magnetic fields are evaluated from the solution for the induced current. The code combines an integral equation for smooth surfaces with one specialized for wires to model a wide range of structures. By use of the method of moment, the integral equations are reduced to a matrix equation. The solution of the matrix equation is then used for evaluation of the currents on wire segments or surface patches. This numerical solution requires a matrix equation of increasing order as the structure size is increased relative to wavelength. Although there are no theoretical size limitations, modeling of structures with dimensions more than several wavelengths is impractical on conventional computers due to limitation in file storage or excessive computing time. This makes the NEC program a very good candidate for conversion to the hypercube. On the hypercube, the fast parallel algorithm and large memory can extend the limits of the NEC many times beyond the conventional computers.

The NEC-2 is a large FORTRAN code containing 77 subroutines and many features such as:

- Computation of scattering or radiation from structures modeled by wires and surface patches
- Effects of perfect or lossy ground
- Modeling thick wires using an extended thin wire kernel
- Modeling of loaded structures (including imperfect conductors)
- Numerical Green's function
- Modeling non-radiating networks.

In the parallel NEC, all features of the sequential NEC are incorporated except lossy ground, numerical Green's function, and non-radiating networks.

B. GENERAL THEORY AND ALGORITHM

The detail description of the theory of the NEC-2 method of moments is given in [4-1]. Here, that theory is discussed in more general terms without any attempt to cover the details.

The NEC Program uses both an electric-field integral equation and a magnetic-field integral equation to model the electromagnetic response of general metallic structures. These integral equations follow the form of an integral representation for the electric field of a volume current distribution or the magnetic field of a surface current distribution respectively. Using these equations and the boundary condition equations on the surface yields a general integral equation where the unknowns are the longitudinal currents on wire segments and two perpendicular components of the surface current on patches. These equations can be expressed in terms of a general linear operator as follows:

$$L\,\tilde{I} = E \qquad (4\text{-}1)$$

where

L is the linear operator consisting of integral and differential operators,

$\tilde{I}$ is the current on the structure, and

E represents the excitation to the system such as a voltage source or an incident field.

In the NEC program, the operator equation (4-1) is solved by a moments method in which the weighting functions, ω_i, are delta functions:

$$\omega_i(\bar{r}) = \delta(\bar{r}-\bar{r}_i) \qquad (4\text{-}2)$$

where

r is the integration variable representing a point on the surface of the structure, and

r_i is the location of the center of a wire segment or a patch.

This choice of the weighting function makes this a point matching technique. The current basis functions are defined separately for wires and patches.

For patches, the basis functions are simple pulse functions

$$V_j(\bar{r}) = F_j\, I_j^P(\bar{r}) \qquad j=1, \ldots, 2M \qquad (4\text{-}3)$$

where

$I_j^P(\bar{r}) = 1$ for $\bar{r}$ on patch j and zero otherwise.

The total current on patches can be expressed as:

$$\tilde{I}^P(\bar{r}) = \sum_{j=1}^{2M} V_j(\bar{r}) = \sum_{j=1}^{2M} F_j \, I_j^P(\bar{r}) \tag{4-4}$$

where

M is the number of patches.

For wires, the basis function is expanded in three terms: a constant, a sine, and a cosine. For segment j the basis function is represented by:

$$I_j^W(\bar{r}) = a_j + b_j \sin k(\bar{r}-\bar{r}_j) + c_j \cos k(\bar{r}-\bar{r}_i),\ j=1,\ \ldots,N \tag{4-5}$$

where

N is the total number of wire segments,

r_j is the location of the center of segment j,

k is the free space wave number, and

a_j, b_j, and c_j are constants

It is assumed that this basis function has a peak on segment j and goes to zero at the other end of the segments connected to segment j (Figure 4.1).

Using the local continuity condition of charge and current, two of the three constants in each basis function are eliminated [4-1] and the total current can be expressed as:

$$\tilde{I}^W(\bar{r}) = \sum_{j=1}^{N} F_j \, I_j^W(\bar{r}) \tag{4-6}$$

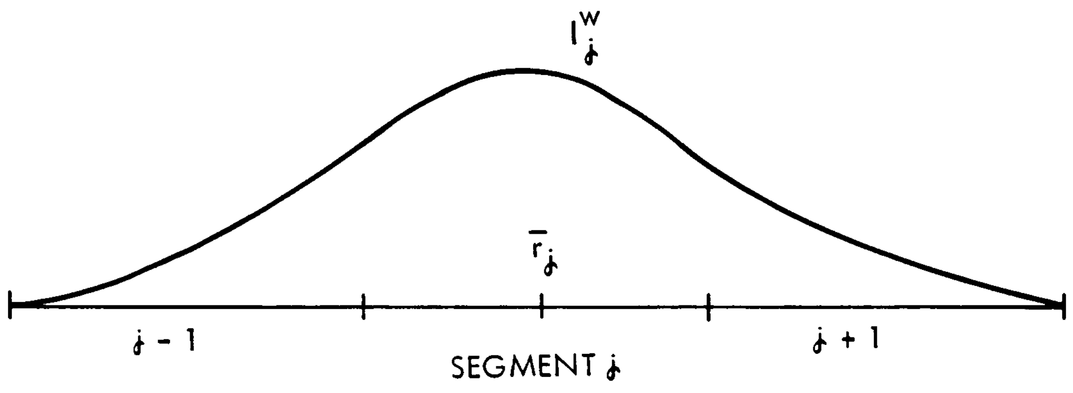


Figure 4.1 The jth basis function for wires.

where

F_i represents the remaining constant.

Combining the current expansion for patches and wires we get:

$$\tilde{I}(\bar{r}) = \sum_{j=1}^{N+2M} F_j \, I_j(\bar{r}) \tag{4-7}$$

where

I_j represents either I_j^P or I_j^W.

Substituting equation (4-7) into equation (4-1) and multiplying both sides by ω_i from equation (4-2) and taking the inner product defined by:

$$<f,g> = \int_S f(\bar{r}) \bullet g(\bar{r}) \, da \tag{4-8}$$

yields the linear system of equations:

$$\sum_{j=1}^{N+2M} F_j <\omega_i, LI_j> = <\omega_i, E>, \tag{4-9}$$

In matrix form equation (4-9) can be written as:

$$[A] \quad [F] \quad = \quad [E] \tag{4-10}$$

where

[A] in general is a (N+2M)*(N+2M) matrix called the interaction matrix

[F] is a (N+2M) * 1 array of basis function amplitudes

[E] is a (N+2M) * 1 array of excitation at the center of all wire segments and patches

In the parallel NEC program, the matrix A is factored to an upper right triangular matrix using the Householder transformation method. Back substitution is then made to compute the basis function amplitudes.

For structures having N wire segments and M surface patches, equation (4-10) can be written as:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} F_w \\ F_p \end{bmatrix} = \begin{bmatrix} E_w \\ H_p \end{bmatrix} \qquad (4\text{-}11)$$

where

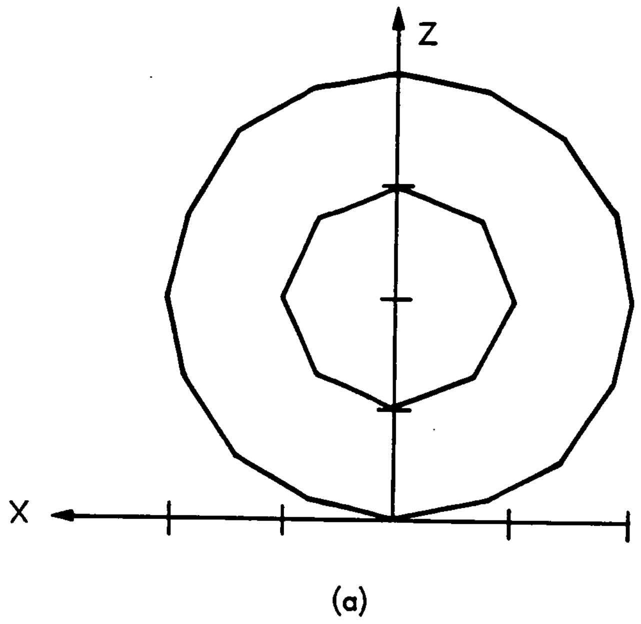
F_w and F_p are basis function amplitudes for wires and patches respectively

E_w and H_p are the electric field at the center of wire segments and magnetic field at the center of surface patches respectively.

The interaction matrix is now divided into 4 submatrices a, b, c, and d. A matrix element a_{ij} in submatrix a represents the electric field at the center of segment i due to the jth segment basis function centered on segment j. A matrix element $d_{k\ell}$ in submatrix d represents a tangential magnetic field component at patch k due to a surface-current pulse on patch ℓ. Matrix elements in submatrices b and c represent electric fields due to surface-current pulses and magnetic fields due to segment basis functions, respectively.

For problems where the structure to be analyzed has cylindrical symmetry, or planes of symmetry such as the one shown in Figure 4.2, the computation time can be reduced significantly. In a case where there are ℓ symmetric cells in the structure, equation (4-10) can be written as:

$$\begin{bmatrix} A_1 & A_2 & A_3 & A_{\ell-1} & A_\ell \\ A_\ell & A_1 & A_2 & A_{\ell-2} & A_{\ell-1} \\ A_{\ell-1} & A & A_1 & A_{\ell-3} & A_{\ell-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A_2 & A_3 & A_4 & A_\ell & A_1 \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ \\ F_\ell \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ \\ E_\ell \end{bmatrix} \qquad (4\text{-}12)$$



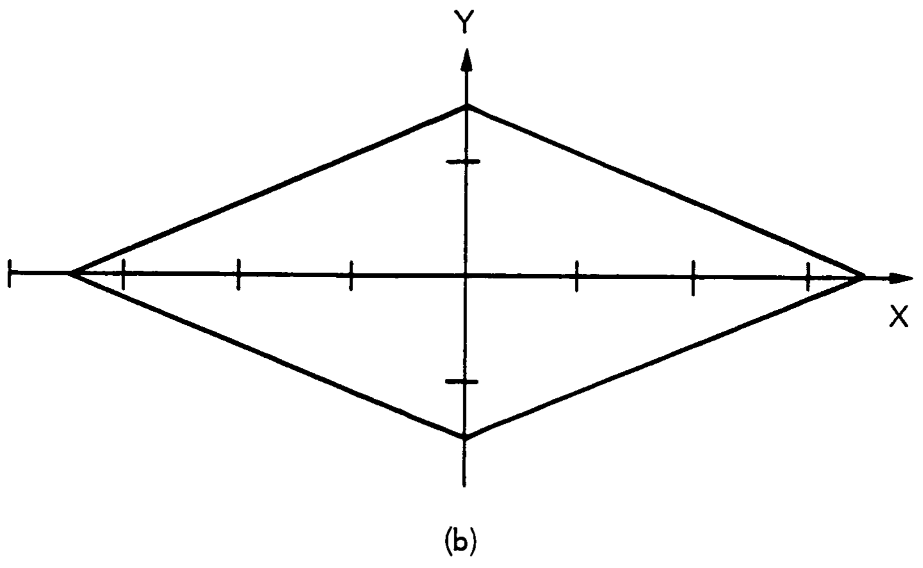


Figure 4.2 Two examples of symmetry: (a) coaxial rings - cylindrical symmetry; (b) rhombic antenna - plane symmetry.

where now each submatrix A_i is of dimension NPEQ = (N'+2M') where N' and M' are the number of wire segments and surface patches in each symmetric cell. This would reduce the time to fill the interaction matrix by a factor $1/\ell$. It can also be shown that by taking discrete Fourier transform of equation (4-12), it can be solved by factoring ℓ matrices of order NPEQ instead of one matrix of order ℓNPEQ. This would reduce the factor time by $1/\ell^2$ and solution (back substitution) time by $1/\ell$. The time to compute Fourier transforms is generally small compared to the time for matrix operations. Symmetry also reduces the number of locations required for matrix storage by $1/\ell$ since only the first row of submatrices need to be stored.

In NEC-2 program, the current of each wire segment can be approximated either by a thin wire kernel or an extended thin wire kernel. In thin wire approximation, the segment current is represented by a current filament. This limits the use of the thin wire models to

$$\frac{\Delta}{a} > 8$$

where

Δ = the length of each segment

a = the radius of each segment.

With extended thin wire approximation, the segment current is represented by a current tube which can be used for models with

$$\frac{\Delta}{a} > 2.$$

Another important feature of the NEC-2 program is its capability to model structures with lumped or distributed loads which includes modeling with lossy wires. This is accomplished by a simple modification to the boundary condition equation. For unloaded structures, the general boundary equation is as follows:

$$\hat{n}(r) * \left[\bar{E}^S(\bar{r}) + \bar{E}^I(\bar{r})\right] = 0 \tag{4-13}$$

where

$\hat{n}$ is the normal vector to the structure.

E_S and E_I are the scattered and incident fields respectively.

For loaded structures, this equation is modified as follows:

$$\hat{n}(\bar{r}) * \left[\bar{E}^S(\bar{r}) + \bar{E}^I(\bar{r})\right] = Z_S(\bar{r}) \left[\hat{n}(\bar{r}) * J_S(\bar{r})\right] \qquad (4\text{-}14)$$

where

Z_S = the surface impedance

J_S = the surface current density

C. PARALLEL DECOMPOSITION OF NEC

1. Structure of the Code

As discussed in Section IV.A, the purpose of the NEC code is to calculate the radiation pattern of an object modeled by wires and patches. The user specifies the incident excitation which will be an incident electromagnetic wave for scattering problems and a voltage source in the object for antenna problems.

The NEC code solves for current I induced in the object by the excitation E. To do this, the unknown current I is expanded in known basis functions with unknown amplitudes F (see Section IV.A). The vector F is found by solving the matrix equation A*F=E where E is the excitation vector and A is the interaction matrix, also described in Section IV.A. After the matrix equation is solved for F, the current I induced in the object is calculated from F and the basis functions. The radiation pattern is then calculated from the induced current I. The structure of the sequential VAX code is illustrated in Figure 4.3.

In the NEC code developed for the Mark III Hypercube, while the basic approach outlined above is unchanged, the actual code structure is changed from that of the original sequential VAX code because of changes in the algorithm used to solve the matrix equation itself and because now, while most of the calculation is done in the hypercube elements (processors), some parts of the calculation are performed in the Control Processor (CP).

The basic structure of the parallel NEC code is shown in Figure 4.4. The structure of the code in the control processor is shown on the left and that of the hypercube, or element, code on the right.

a. Input. The input section of the code is performed in the Control Processor. This section includes reading the various input data cards from a file, creating the wires and patches from the geometry input cards, setting up various parameters regarding the excitation vector, and determining from the input cards what output is to be calculated.

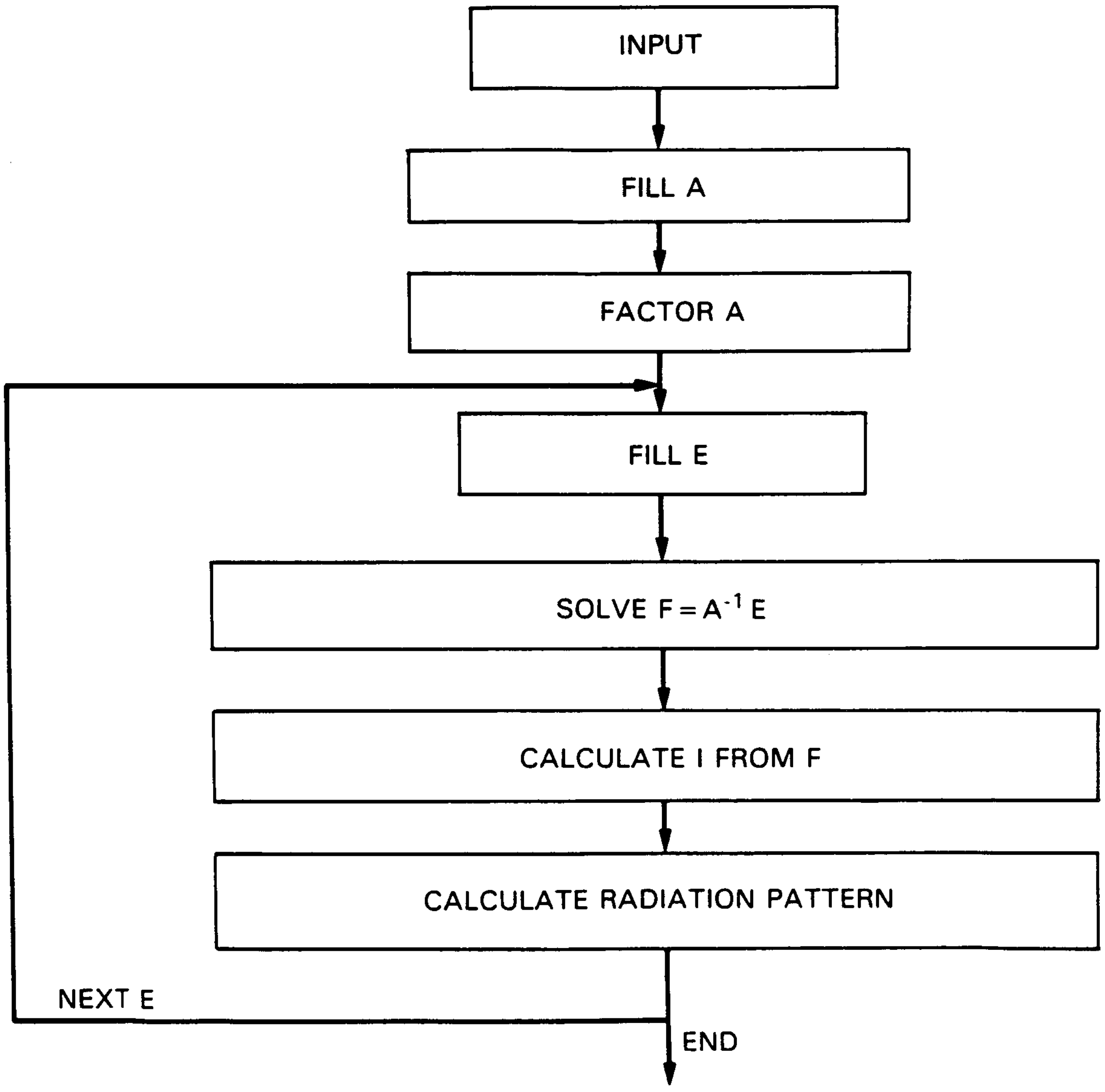


Figure 4.3 Structure of the sequential NEC-2 program.

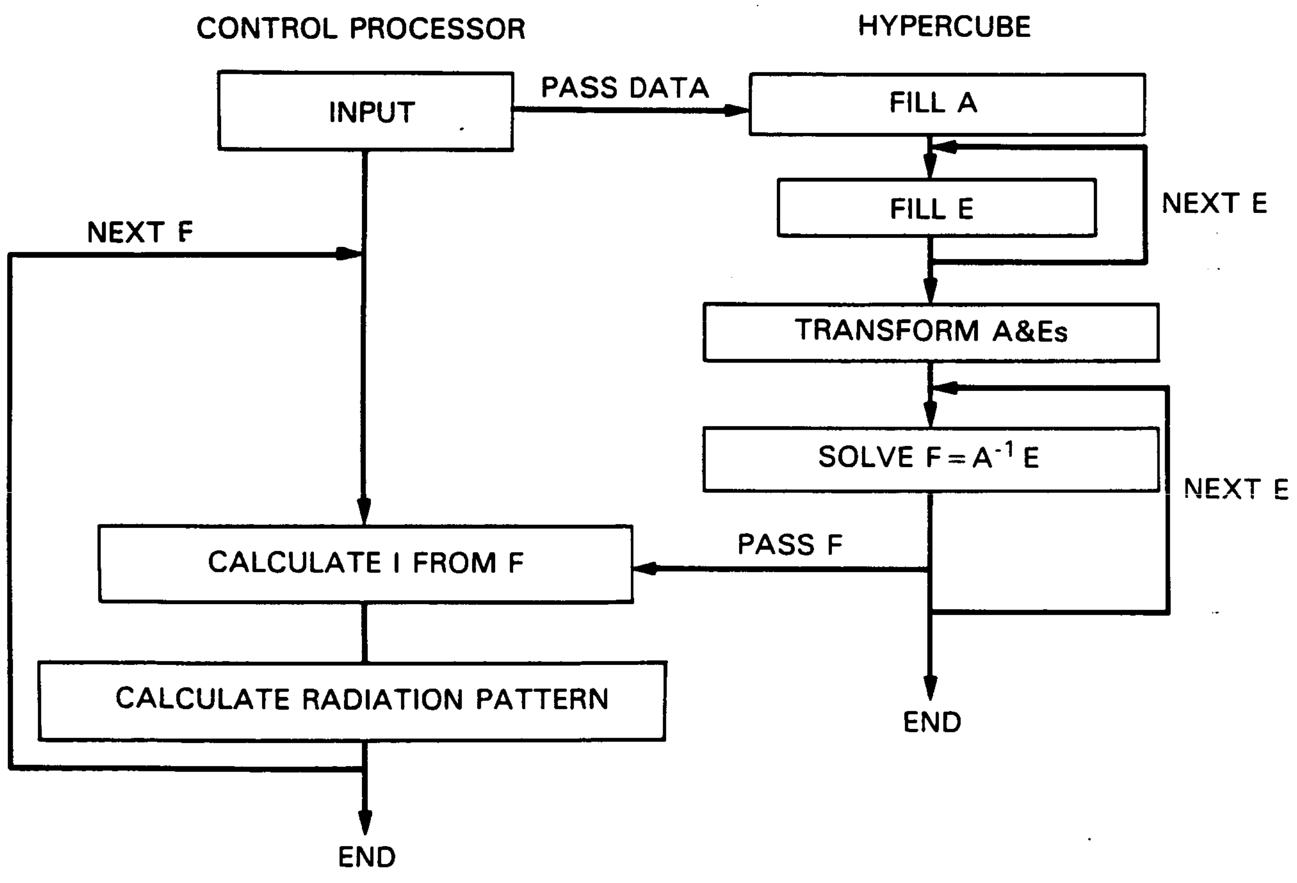


Figure 4.4 Structure of parallel NEC program.

First, the input data relating to setting up and solving the matrix equation is read and processed. This includes all of the geometry and excitation information. If the scattered radiation pattern is to be calculated for an electromagnetic wave incident from several different angles, NEC must solve the matrix equation A*F=E for the same A matrix but several different E vectors, corresponding to the different angles of incidence. In this case, the parameters needed for filling all of these E's are calculated at the start in the CP in the new subroutine ETMFILL. All of the information needed for filling the A matrix and the E vectors is then passed to the hypercube via the CP subroutine CPCOMM and the hypercube main program NECELT. At this point, the CP sits idle until it receives the solution vector F for the first excitation vector E.

b. Matrix Fill. After receiving the initialization data from the CP, the element code first calls the subroutine CMSETN which handles the fill of the interaction matrix A. The parallel fill of this matrix is described in Section IV.C.2 below. If there is symmetry in the modelled object, NEC can make use of this to considerably reduce the order of the matrix equation to be solved. For such symmetric cases, the discrete Fourier transform of the matrix is also done in parallel at fill time in the subroutine CMSETN.

c. Excitation Fill. After the matrix fill, the excitation E is calculated via the subroutine ELTETM, described in Section IV.C.3. If the matrix equation is to be solved for multiple right-hand sides, all E vectors are filled at this time through multiple calls to this subroutine.

For cases with ℓ symmetric subsections, the ℓ-point discrete Fourier transform of the E's is next performed by the subroutine ELTDFT. Multiple calls to this subroutine are made for the case of multiple excitation vectors.

The E's and their discrete Fourier transforms are calculated here before the Householder transformation is called because they also must be transformed. Since the Householder transformation is one of the most time consuming portions of the code, all E's (and their discrete Fourier transforms for symmetry cases) must be available at this stage so that the transformation of A need only be done once. Note that the code structure here is quite different from the original sequential NEC.

d. Householder Transformation. The Householder transformation of the A matrix and all the E vectors is performed after the fill of E and A. The parallel Householder algorithm is described in Section IV.C.4. It transforms the original matrix to an upper triangular matrix so that the resulting matrix equation can be solved by back substitution.

The main element program NECELT calls the subroutine FACTRSN which then performs the transformations of A. For cases with ℓ-fold symmetry, the transformation is done separately on the ℓ symmetric sections of A and of all the E's. Thus FACTRSN makes ℓ calls to the Householder routine HHNFACTR. On each call, a symmetric subsection of A is transformed along with the corresponding symmetric sub-vectors of all the excitation vectors, E.

Thus, for multiple right-hand sides, the Householder transformation of A (or each of its symmetric sub-sections) is done only once.

e. Matrix Equation Solution. After the Householder transformation of the A matrix and the E vectors has been completed, the transformed matrix equation is solved by back substitution for one E vector at a time. The back substitution method is described in Section IV.C.4.

The solution is done in the subroutine BSOLVE which solves the matrix equation separately for each of the ℓ symmetry subsections. BSOLVE is called by the subroutine ELTSOLN. For symmetric cases, ELTSOLN performs the inverse discrete Fourier transform of the solution vector F.

After the solution vector F is found for a particular E, this vector is passed back to the CP. For cases with multiple right-hand sides, the hypercube then makes another call to BSOLVE to solve the transformed matrix equation for the next E while the CP is processing the previous solution vector F.

After the hypercube finds and passes back the solution vector F for the last excitation E, the hypercube code terminates.

f. Current Calculation and Output. When the CP receives its first solution vector F, the induced current I is calculated sequentially from F by the subroutine CABC. The CP then calculates whatever output was specified in the input data, e.g., near fields and far field radiation patterns, and writes the output to a file.

For cases with multiple right-hand sides, the CP then sits idle until it receives the next solution vector F. When the output from all excitations has been found, the CP code terminates.

2. Interaction Matrix Fill

The interaction matrix A is shown in terms of its four submatrices a, b, c, and d in equation (4-11). In the NEC program, the elements of these submatrices are computed by subroutines CMWW (wire to wire interaction), CMSW (surface to wire interaction), CMWS (wire to surface interaction), and CMSS (surface to surface interaction), respectively. An element A_{ij} of the interaction matrix is computed by evaluating the electric field or magnetic field at the center of an observation segment (a segment can be a wire segment or a surface patch) due to the basis function centered on another segment called the source segment. Figure 4.5 graphically depicts this process for wire to wire interaction. Computation of each element of the interaction matrix is therefore accomplished in three steps:

(1) Defining a source segment (identified by index j) and computing some data related to that segment

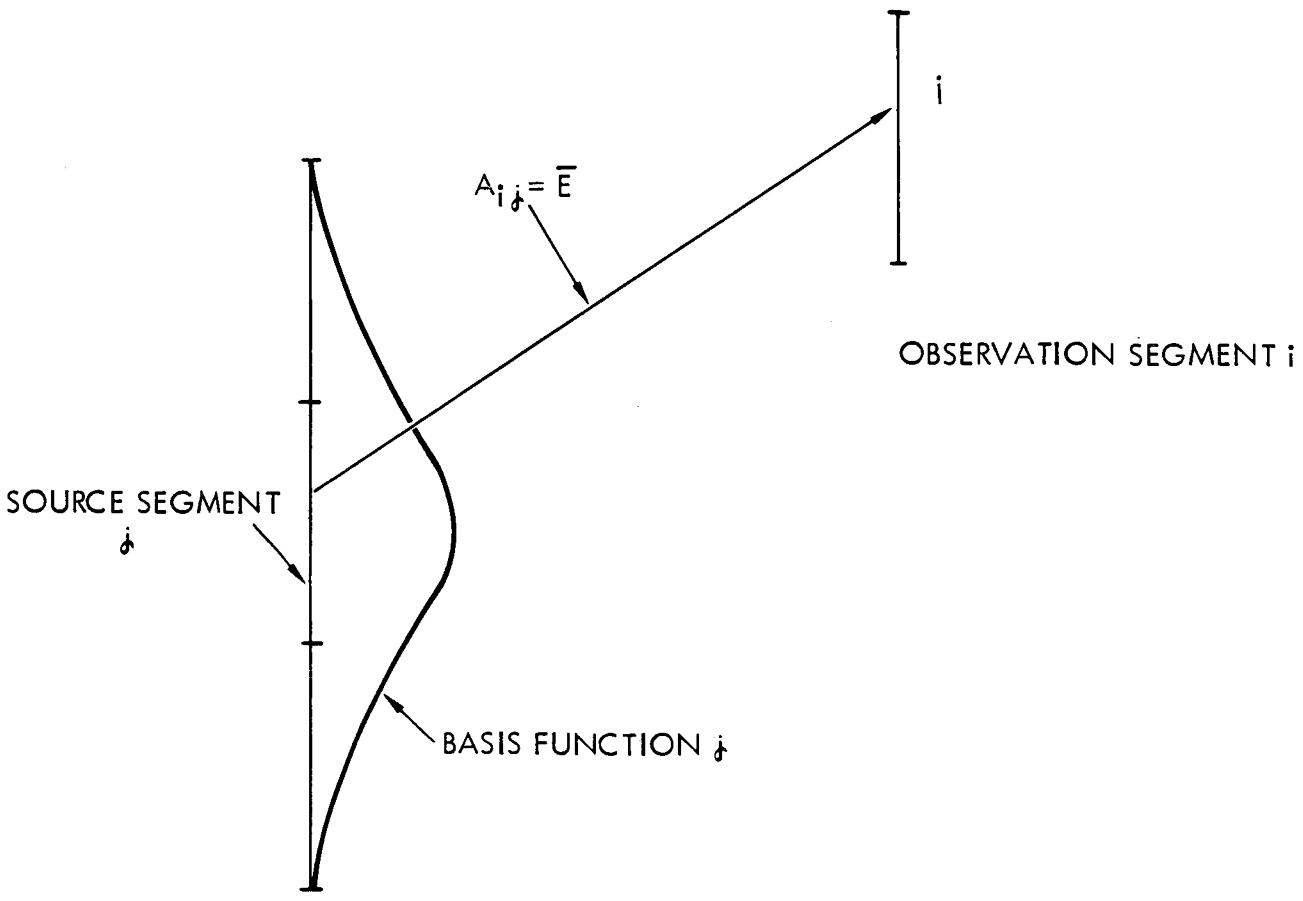


Figure 4.5 Interaction of wire segments i and j.

(2) Defining an observation segment (identified by index i) and computing data related to that segment

(3) Finally, evaluating either the electric field or magnetic field using the information obtained in the previous steps.

The order in which the first two steps are taken differs depending on the submatrix that is being filled.

In the sequential NEC Program, the interaction matrix is filled inside a double DO-loop with index variables i and j, where i and j go from 1 to N_T (total number of segments). Figure 4.6 shows a block diagram for the sequential code. In the first DO-loop a source segment j is picked and the current expansion functions on that segment and adjacent segments are computed. In the second DO-loop an observation segment i is picked and the electric field or magnetic field on that segment is computed depending on if it is a wire segment or a surface patch. Consequently, in row i, column j and columns corresponding to segments connected to segment j are filled.

In the parallel NEC program several processors are used to fill the interaction matrix. Therefore, each processor will be responsible for filling a part of the interaction matrix. To minimize communication between processors, each processor is made responsible for filling a number of rows of the interaction matrix entirely. The rows are assigned to the hypercube processors according to the chart shown in Figure 4.7, that is for a cube with np processors, processor n will compute rows n, $n+n_p$, $n+2*n_p$, ... etc. With this distribution scheme, if the number of rows are equal to a multiple of the number of processors, then all processors will fill the same number of rows. Otherwise, some processors fill one row more than the rest.

Two parallel codes are developed for matrix fill. In the first code, called the source loop sequential code (SLSC), only the observation loop is computed in a parallel manner. The data in the source loop is computed the same as in the sequential NEC program and therefore some redundant information is processed in the cube causing a reduction in the efficiency. On the other hand, with this algorithm, there is no need for the cube processors to communicate with each other which helps increase the efficiency. A block diagram of the code with sequential source loop is shown in Figure 4-8. Here, the program in each processor (or node) is basically the same as in the sequential code except in the observation loop where the index i takes on different values depending on the processor in which the code is running. Therefore each processor fills only the rows which are assigned to it.

The second code, called source loop parallel (SLPC), is an extension of the source loop sequential program. A block diagram of this code is shown in Figure 4.9. Here, the observation loop is computed like the SLSC but to avoid processing redundant information, the source loop is made parallel as well as the observation loop. This is implemented for wires only since the source information for patches can be computed faster than it can be communicated at all times. In this program, each processor will compute the source data for a few segments only. For example, node m computes source data for segments j=m, $m+n_p$, $m+2*n_p$, ... Therefore, the data for all source segments is available but divided among all the nodes. Hence, for a processor to obtain the

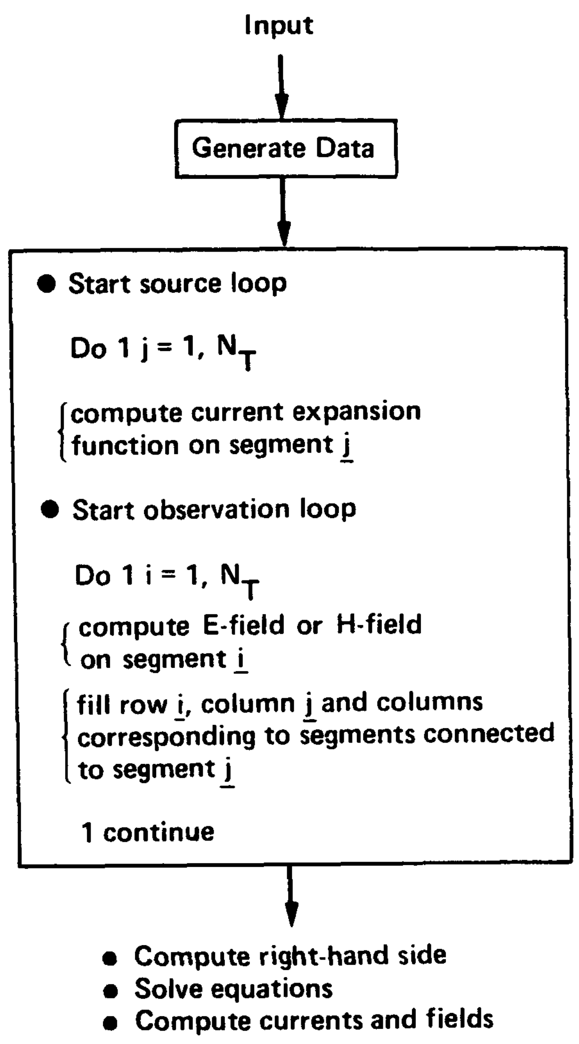

Input
Generate Data
● Start source loop
Do 1 j = 1, N_T
compute current expansion function on segment j
● Start observation loop
Do 1 i = 1, N_T
compute E-field or H-field on segment i
fill row i, column j and columns corresponding to segments connected to segment j
1 continue
● Compute right-hand side
● Solve equations
● Compute currents and fields

Figure 4.6 Sequential interaction matrix fill.

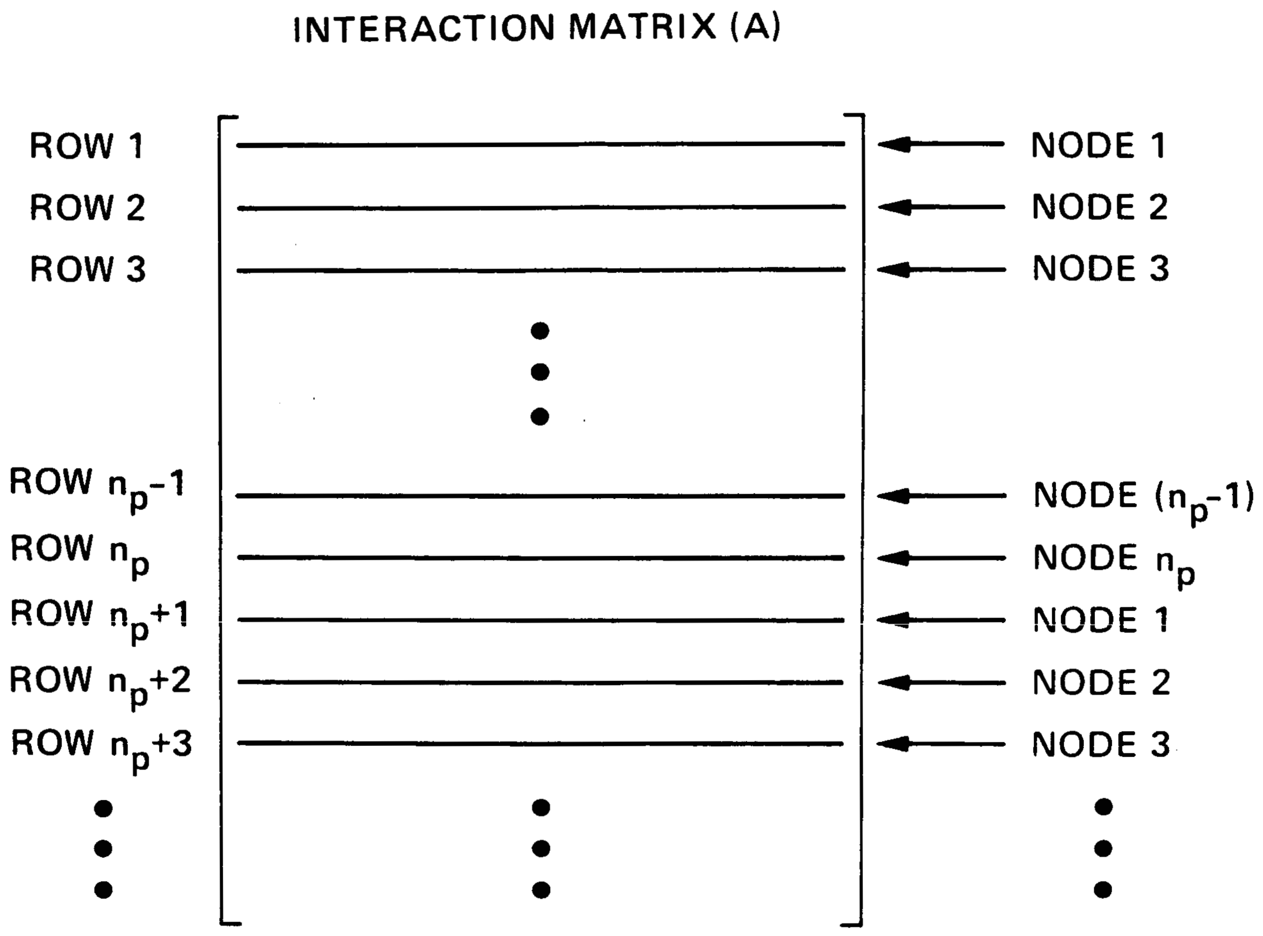


Figure 4.7 Assignment of rows to hypercube processors.

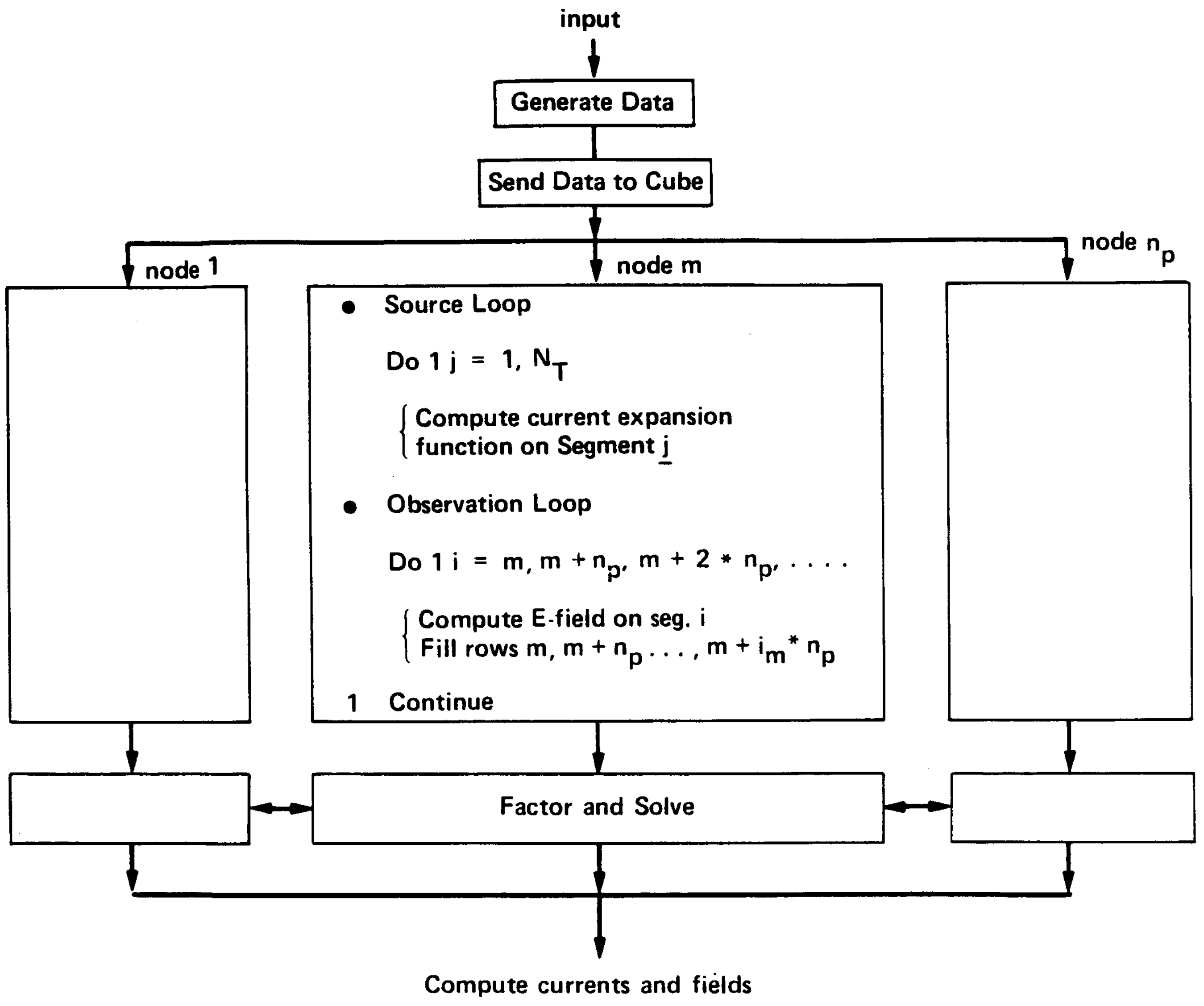

input
Generate Data
Send Data to Cube
node 1
node m
node n_p
● Source Loop
Do 1 j = 1, N_T
Compute current expansion function on Segment j
● Observation Loop
Do 1 i = m, m + n_p, m + 2 * n_p, ...
Compute E-field on seg. i
Fill rows m, m + n_p . . . , m + i_m * n_p
1 Continue
Factor and Solve
Compute currents and fields

Figure 4.8 Parallel interaction matrix fill program - source loop sequential.

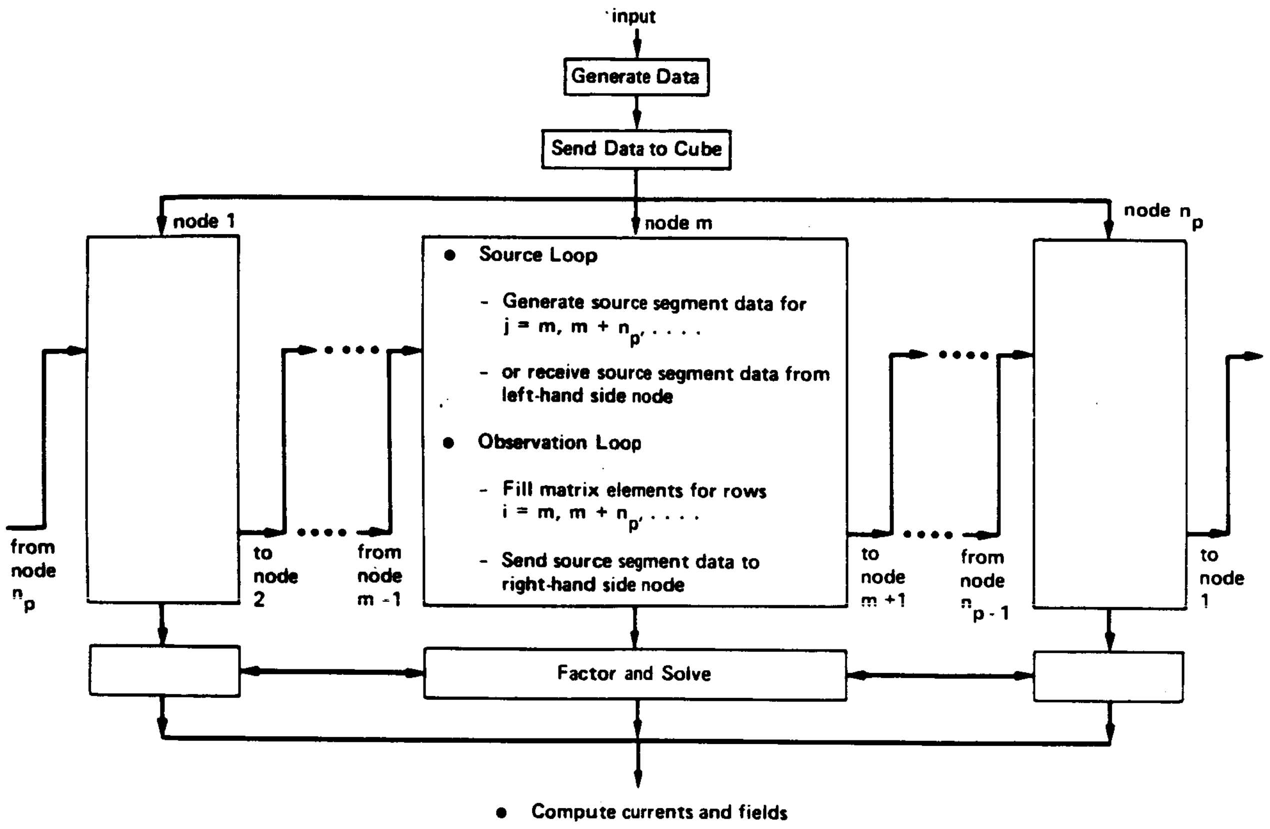

input
Generate Data
Send Data to Cube
node 1
node m
node n_p
● Source Loop
- Generate source segment data for $j = m, m + n_p, \ldots$
- or receive source segment data from left-hand side node
● Observation Loop
- Fill matrix elements for rows $i = m, m + n_p, \ldots$
- Send source segment data to right-hand side node
from node n_p
to node 2
from node m -1
to node m +1
from node n_{p-1}
to node 1
Factor and Solve
● Compute currents and fields

Figure 4.9 Parallel interaction matrix fill program - source loop parallel.

source data for all segments, it has to receive some of that data through communication channels from other nodes. For this purpose, the nodes are mapped into a one dimensional periodic lattice (or placed on a ring) (Figure 4.10). With this arrangement, each node can send or receive information from its two neighboring processors only. The information flow is set to be counterclockwise and therefore, node m will receive data from node m-1 but sends data to node m+1.

To better describe the source loop parallel algorithm, consider the example shown on Figure 4.11. It is assumed that a problem with 6 segments is being computed by a 4-node hypercube. Now, let's see what happens in node 1 at different times after the source loop is started at time t_0:

- At time t_1, node 1 computes source data for segment j=1.
- At time t_2 it sends data for j=1 to node 2 and receives data for j=4 from node 4.
- At time t_3 it sends data for j=4 to node 2 and receives data for j=3 from node 4.
- At time t_4 it sends data for j=3 to node 2 and receives data for j=2 from node 4.
- At time t_5 all nodes including node 1 compute source data for segments j=5,6.

This means that if there is a remainder in the division of the number of segments by the number of processors, the data for the remaining segments is computed sequentially. Figure 4.11 explains the overall flow of information for the above example.

3. Parallel Fill of the Excitation Vector

NEC solves the matrix equation A*F=E where the excitation vector E and the solution vector F have N_T=N+2*M elements for an object modeled with N wires and M patches. For objects with no symmetry, the interaction matrix A is N_T*N_T. For an object with ℓ symmetric subsection, A has N_T columns and NPEQ rows where NPEQ=N_T/ℓ as discussed in Section IV.A. When the solution vector F is found, it is written over the E vector and so the parallel decomposition of E also determines the parallel decomposition of F.

The parallel decomposition of the E fill is determined by the parallel decomposition of the matrix fill. For cases with no symmetry, the only restriction on E is that the i-th element of E be in the same processor as the i-th row of A so that the back-substitution algorithm can proceed without the necessity of fetching the needed element of E from a different processor. Thus, the elements of E are "dealt out" to the processors as the rows of A were (see IV.C.2): The first element goes to the first processor, the second element to the second processor, the third element to the third processor etc., returning to the first processor after the first n_p elements have been dealt, where n_p is the number of processors. The deal proceeds until all

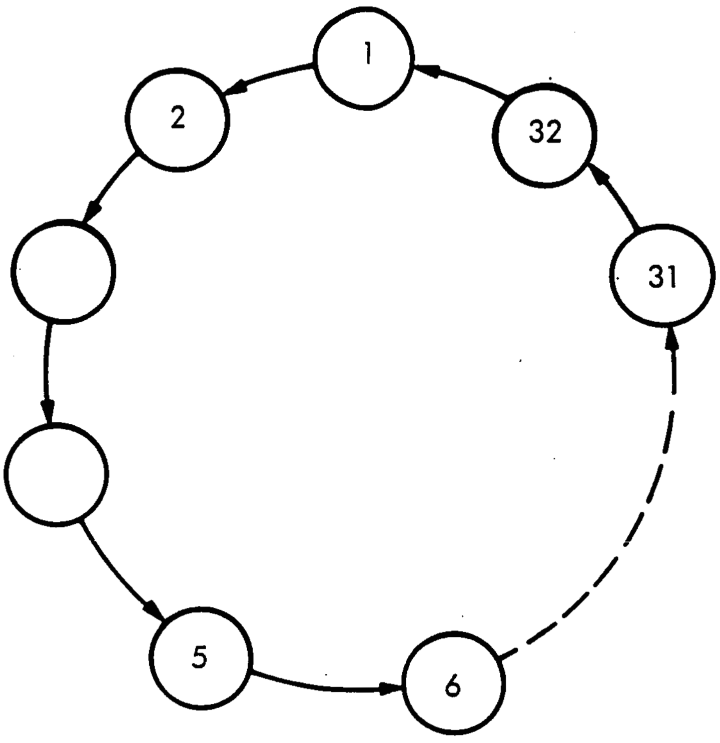
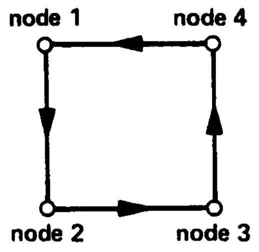


Figure 4.10 Node arrangement for source loop parallel program in a 32-node hypercube.

- Assume number of segments to be:

  N = 6 ; J = 1, 6



- Source segment data is either computed in nodes or transferred to nodes in the following sequence in node $\begin{cases}1\\2\\3\\4\end{cases}$

  – Time t_0 start source loop

  – Time t_1 compute source segment data for J = $\begin{cases}1\\2\\3\\4\end{cases}$

  – Time t_2 send data for J = $\begin{cases}1\\2\\3\\4\end{cases}$ to node $\begin{cases}2\\3\\4\\1\end{cases}$ Receive data for J = $\begin{cases}4\\1\\2\\3\end{cases}$ from node $\begin{cases}4\\1\\2\\3\end{cases}$

  – Time t_3 send data for J = $\begin{cases}4\\1\\2\\3\end{cases}$ to node $\begin{cases}2\\3\\4\\1\end{cases}$ Receive data for J = $\begin{cases}3\\4\\1\\2\end{cases}$ from node $\begin{cases}4\\1\\2\\3\end{cases}$

  – Time t_4 send data for J = $\begin{cases}3\\4\\1\\2\end{cases}$ to node $\begin{cases}2\\3\\4\\1\end{cases}$ Receive data for J = $\begin{cases}2\\3\\4\\1\end{cases}$ from node $\begin{cases}4\\1\\2\\3\end{cases}$

  – Time t_5 compute source segment data for J = 5 and 6

Figure 4.11 Example - parallel source loop algorithm on 4 nodes.

N_T elements of E have been dealt. Thus, the first processor ends with elements 1, $1+n_p$, $1+2*n_p$,...; the second processor with 2, $2+n_p$, $2+2*n_p$, ..., etc.

For cases with symmetry, the "deal" of E changes somewhat, although the parallel decomposition is still determined by the decomposition of A and the requirements of the back substitution algorithm. As discussed in Section IV.C., the code now solves ℓ matrix equations of order NPEQ=N_T/ℓ where ℓ is the number of symmetric submatrices of A. In these cases, E is also composed of ℓ symmetric subvectors of length NPEQ. The NPEQ rows of A have been dealt out as in the case with no symmetry. Now, the ℓ symmetric subvectors are dealt out separately, always assigning the first element of the subvector to the first processor, the second element of the subvector to the second processor, etc., until the NPEQ elements of the subvector have been dealt (see Figure 4.12). In the end, the corresponding elements of each subvector of E are in the same processor; e.g., elements 2, 2+NPEQ, 2+2*NPEQ are in the second processor. The number of elements in each processor, then, is always a multiple of ℓ.

For cases with symmetry, E, as well as the interaction matrix A, is expanded in a discrete Fourier series immediately after it is filled. At solution time, the elements of A, E and F are the coefficients of this expansion. Although the parallel decomposition of E was determined by the decomposition of A, this decomposition has the added feature that all of the elements of E needed to perform the discrete Fourier transform have been assigned to the same processor. Thus, no internode communication was necessary to perform the discrete Fourier transform of E and, likewise, to perform the inverse discrete Fourier transform of the solution vector F.

4. Factorization and Solution

The Householder transformation is the technique used for the inversion of the interaction matrix [4-2]. This inversion method is well-suited to a parallel implementation because it operates on columns of the matrix where each column's set of computations is independent of the computations performed in the other columns (Figure 4.13). As the inversion progresses, the matrix is factored by a series of orthogonal transformations producing an upper right triangular factored matrix and a lower left triangle containing zeros (Figure 4.14). The right hand side excitation vectors undergo the same series of transformations at the same time that the interaction matrix is transformed. In this way there is no need to store (or, for that matter, even explicitly compute) the transformation matrix.

The factorization subroutine is written in the C programming language rather than FORTRAN which is used for most of the parallel NEC code. From benchmark tests of the relative speed between C and FORTRAN, C has been found to perform about 15% faster than FORTRAN for the same instructions when the C program utilizes pointers for array access and uses register variables. Therefore, since the factorization portion of the NEC-2 code is potentially the most time consuming and the parallel factorization represents new code, it has been decided to implement this factorization in C.

$$(A^1\ A^2\ A^3)\ \text{AND}\ \begin{pmatrix}E^1\\E^2\\E^3\end{pmatrix}$$

$$\begin{array}{l|cccc|}
\text{PROCESSOR 1} & A_{1,1} & A_{1,2} & \bullet\ \bullet\ \bullet & A_{1,N}\\
\text{PROCESSOR 2} & A_{2,1} & A_{2,2} & \bullet\ \bullet\ \bullet & A_{2,N}\\
\bullet & \bullet & \bullet & \ddots & \bullet\\
\bullet & \bullet & \bullet & & \bullet\\
\bullet & \bullet & \bullet & & \bullet\\
 & A_{\frac{N}{3},1} & A_{\frac{N}{3},2} & & A_{\frac{N}{3},N}
\end{array}$$

$$\begin{array}{|c|l}
E_1 & \text{PROCESSOR 1}\\
E_2 & \text{PROCESSOR 2}\\
\bullet & \bullet\\
\bullet & \bullet\\
\bullet & \bullet\\
E_{\frac{N}{3}} & \\
E_{\frac{N}{3}+1} & \text{PROCESSOR 1}\\
E_{\frac{N}{3}+2} & \text{PROCESSOR 2}\\
\bullet & \bullet\\
\bullet & \bullet\\
\bullet & \bullet\\
E_{\frac{2N}{3}} & \\
E_{\frac{2N}{3}+1} & \text{PROCESSOR 1}\\
E_{\frac{2N}{3}+2} & \text{PROCESSOR 2}\\
\bullet & \bullet\\
\bullet & \bullet\\
\bullet & \bullet\\
E_N &
\end{array}$$

Figure 4.12 Assignment of processors for an object with 3-fold symmetry and N segments.

Performed in parallel:

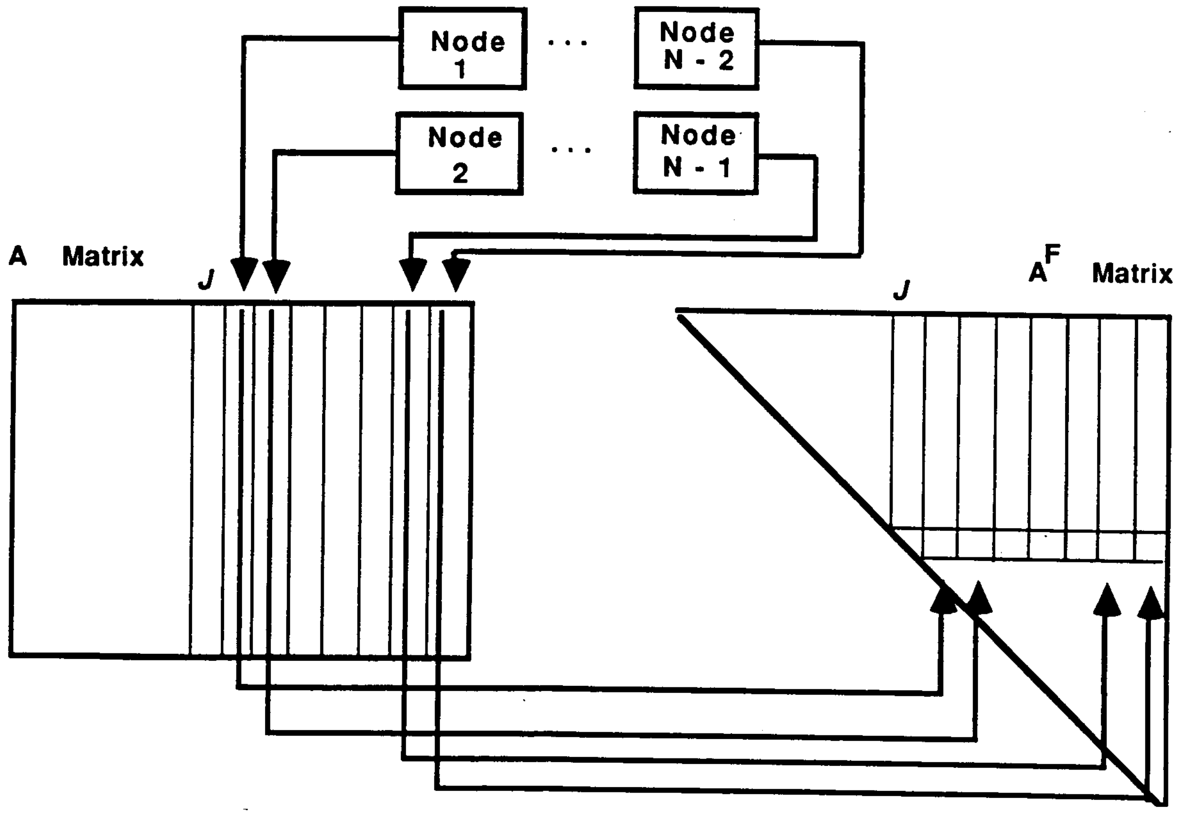
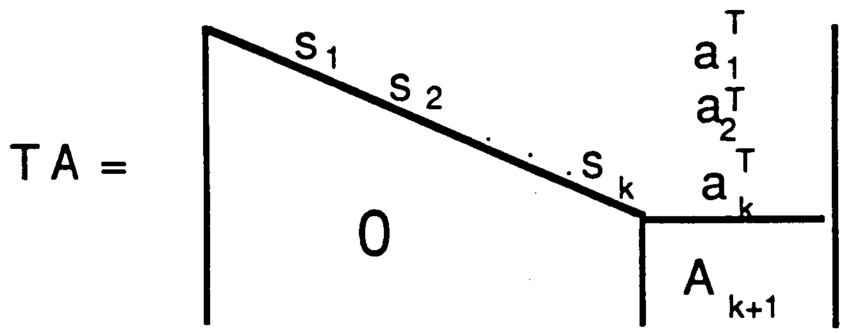


Figure 4.13 The Jth transformation.

For matrix A (m,n), the Householder Transformation is:



where

T is the orthogonal transform matrix

s_k are the diagonal elements of the transformed matrix

a_k are the transformed elements of the A matrix

Figure 4.14 Householder transformation.

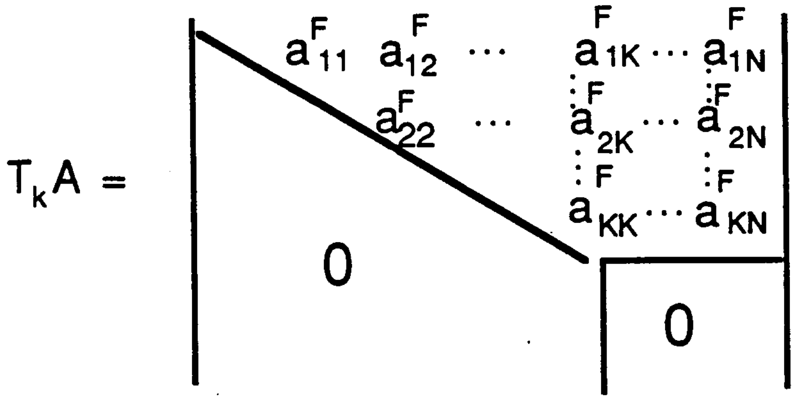
a. Distribution of Data on Input. The interaction matrix enters the factorization subroutine HHNFACTR having been distributed by rows to the processors in the subroutine CMSETN. Since the Householder transformation operates on columns of the matrix, the first step in the parallel implementation is to transpose the distribution of the data; i.e., the processors become responsible for columns rather than rows. This data redistribution is accomplished by utilizing a ring mapping of the processors. Each processor loads all of its row data into a communication buffer and transmits it to its left-hand neighbor while receiving data in a similar buffer from its right. Each processor then picks off column data destined for its processor. This sequence continues for n_p-1 steps (where n_p is the number of processors being used) until each processor has all of its column data. This initial shuffling is more advantageous than intermittent shuffling because all processors perform the transposition in lockstep and with the same amount of data. In the Crystalline Operating System (CrOS) it is important to minimize the communication synchronization time in order to optimize the parallel algorithm's performance. Columns are assigned to processors in the same fashion as playing cards are dealt from a deck. Therefore all processors have no more than one difference in the total number of columns which they will process. The data within a column is stored contiguously in order to minimize the distance of memory accesses and computation of array offsets.

The excitation vector components calculated within the subroutine ELTETM are distributed to correspond to the distribution of the row components in the interaction matrix. Since each of these vectors will be treated as a column in the transformation, each right-hand side vector is assembled in the assigned processor. The assignments are determined by continuing the dealing method for column assignments thereby again assuring optimum load balance.

b. Factorization. The Householder transformation consists of NPEQ transformations (where NPEQ is the number of rows in the interaction matrix). For the first transformation, the data in the first column is distributed to all processors via a BROADCAST call. Each processor uses this column to determine the new elements for its columns as well as those elements which it contributes to the first row of the factored matrix. At the conclusion of the first transformation, the interaction matrix contains new columns for columns two through NPEQ; column one will no longer be active. In the second transformation, the processor which has column two distributes it to all other processors. Again each processor computes the new elements for its interaction matrix active columns as well as the elements of the second row of the factored matrix for which it is responsible. The transformation progresses, each time with one less column active, until NPEQ transformations have been performed and the complete upper right triangular matrix is constructed (Figure 4.15).

In order to economize on storage the newly computed factored matrix row elements are overwritten on the inactive portion of the transformed interaction matrix. To do so, since the elements are distributed among the processors at the end of a transformation step, this data is combined and assembled in the processor which will eventually do the solution for that

The factored matrix A^F after the kth Householder Transformation:



$T_kA =$
a_{11}^F a_{12}^F ⋯ a_{1K}^F ⋯ a_{1N}^F
a_{22}^F ⋯ a_{2K}^F ⋯ a_{2N}^F
a_{KK}^F ⋯ a_{KN}^F
0
0

To minimize storage, the A matrix columns are stored as rows. Then as factoring progresses the A^T and A^F are merged:

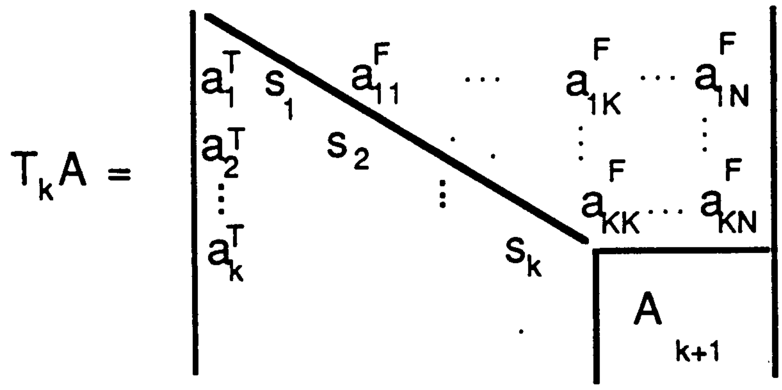


Figure 4.15 Householder transformation after kth transformation. A^T is the working matrix which is discarded after the factorization. A^F is the newly factored upper right triangular matrix.

row. This row assignment matches the earlier row assignment in CMSETN. At the end of each transformation the newly calculated transformed elements of the excitation vectors are also replaced in the source processors again matching the earlier row assignments from ELTETM. At the conclusion of the factorization subroutine HHNFACTR, the data is in position for the back substitution.

c. The Solution. Back substitution is used to calculate the solution for each right-hand side excitation vector. Each processor calculates the solution for the rows for which it is responsible. At the conclusion of each row, the solution elements are broadcast to the other processors. These solutions become multiplicands for subsequent steps of the solution. At this time the solution performs nearly sequentially, but over the distributed data. Once the Mercury asynchronous operating system (which will permit queuing of asynchronous messages) extends to the FORTRAN or hybrid codes such as parallel NEC, we can anticipate a modest speedup over the current back substitution method.

D. NUMERICAL RESULTS

The numerical results obtained from the parallel NEC are extensively checked with results from the original NEC-2 (sequential NEC) and an excellent agreement is seen for every case. The result for three problems, where wires or patches, either separate or combined, are used is discussed here. Also examples are shown for validation of the code when extended thin wire approximations or loading are employed.

1. Monopole on a Pedestal Over Perfect Ground

This problem is modeled by as many as 290 wire segments. The pedestal is modeled by several radial 2-wire sections (or ribs) as shown in Figure 4.16. Each wire, in turn, is divided into several segments. A quarter wave monopole is placed at the center of the pedestal and is fed by a voltage source at the bottom. The far field radiation pattern of this monopole, in Plane $\phi=0$, is shown in Figure 4.16 where ϕ and θ are the standard azimuthal and polar angles in the spherical coordinate system. The solid line is from the sequential code while the dots show the results from the parallel code. For all practical purposes perfect agreement exists between the two codes. Note the bend in the pattern which is due to the presence of the pedestal. By increasing the number of ribs in the wire model, the effect of the pedestal will become more significant.

2. Scattering of a Plane Wave by a Conducting Sphere

The conducting sphere is modeled by patches only. In the example shown on Figure 4.17, the sphere is modeled by 80 patches without taking advantage of the symmetry. The incident field is a uniform plane wave traveling in -x direction and is polarized in -z direction. The far field scattering pattern of the sphere, for plane $\phi=0$, is shown in Figure 4.17.

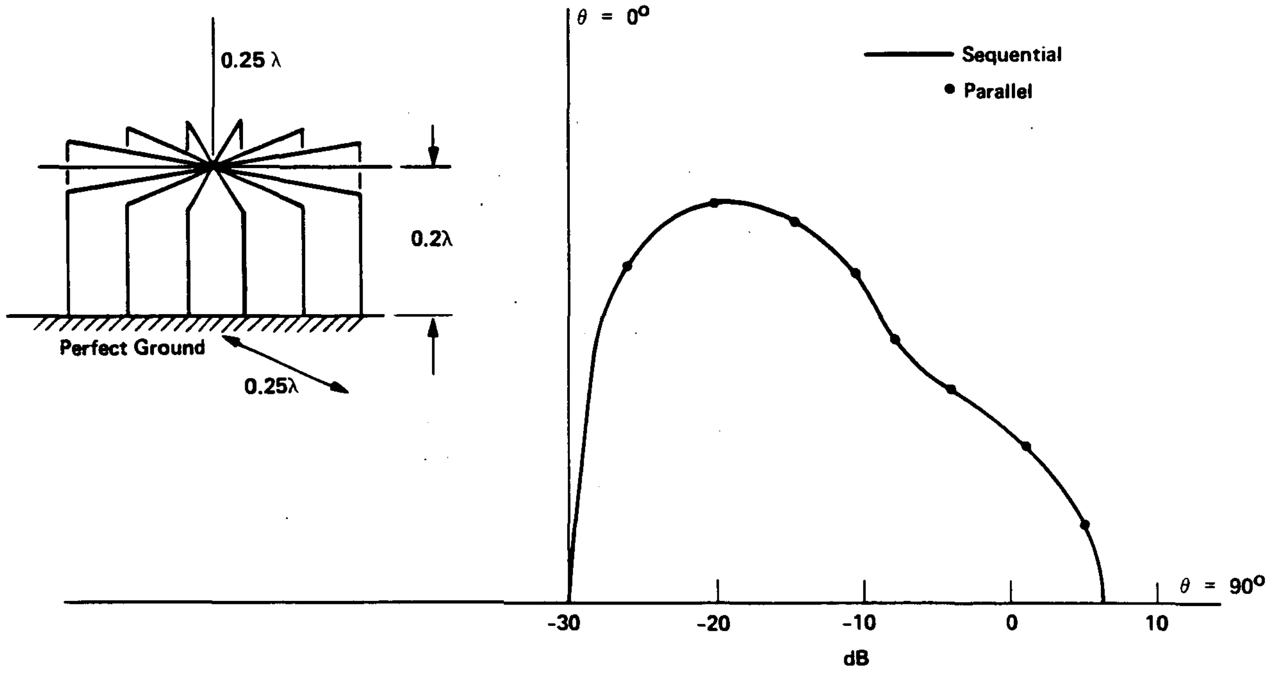


Figure 4.16 Radiation pattern of a quarter wave monopole on a pedestal over perfect ground; $\phi=0$ cut. The structure is modeled by 130 wire segments.

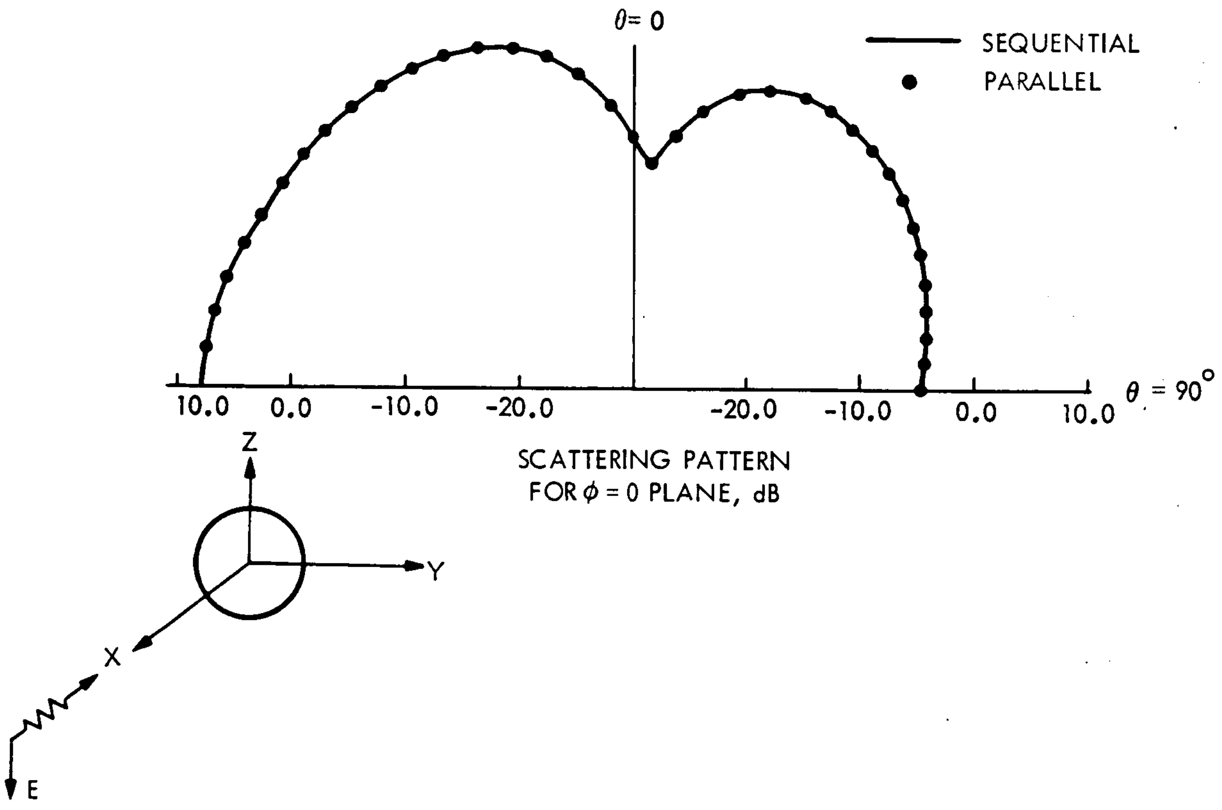


Figure 4.17 Scattering pattern of a sphere with ka = 3.0 in ϕ=0 plane. Incident field is a plane wave traveling in -x direction. The sphere is modeled by 80 patches.

The solid line shows the result from the sequential NEC code, while the dots show the result from the parallel code. Like the previous example very good agreement exists here. For this example ka = 3.0

where

$$k = \frac{2\pi}{\lambda}$$

λ = the plane wave wavelength

a = the radius of the sphere.

3. T Antenna On a Conducting Box Over Perfect Ground

This problem is modeled by both patches and wires. The box is modeled by 12 patches and the T antenna by 8 wire segments. This is the same as Example 4 in the NEC manual [4-1]. The T antenna is fed by a voltage source at the bottom. The radiation pattern of this antenna, for ϕ=0 plane, is shown in Figure 4.18. The solid line is the result from the sequential NEC and the dots show the solution from the parallel NEC.

4. Extended Thin Wire and Loading

To show the effect of using the extended thin wire kernel and loading, the monopole on a pedestal of Section IV.D.1 is employed. The structure is modeled by 70 segments. The radiation pattern of the monopole, for ϕ=0 plane, is shown in Figure 4.19. Here, the solid line is for the case where thin perfectly conducting wires are used, dots show the result for thin aluminum wire (loading), and Δ marks show the solution for the case where thick wire is used with extended thin wire kernel. All these fields are calculated by the parallel code and excellent agreement with the sequential code is checked for every case. It can be noted that the effect of the pedestal on the pattern is less significant when a fewer number of ribs are used (10 ribs here compared to 15 ribs in the example shown in Figure 4.16). Also, use of extended thin wire, even with a lower number of ribs, models the pedestal much more realistically than the thin wire kernel modeling of Section IV.D.1. This is evident from the strong interaction of the fields with the pedestal shown by Δ marks in Figure 4.19.

E. PERFORMANCE

To evaluate the performance of the NEC parallel program, the CPU time taken to run this code on the Mark III Hypercube is compared with that on a VAX 11/750 computer. Several parameters, such as the time taken to fill or factor the interaction matrix, and the speedup factor for a different number of nodes used in the hypercube or a different number of segments used in the structure model, are analyzed.

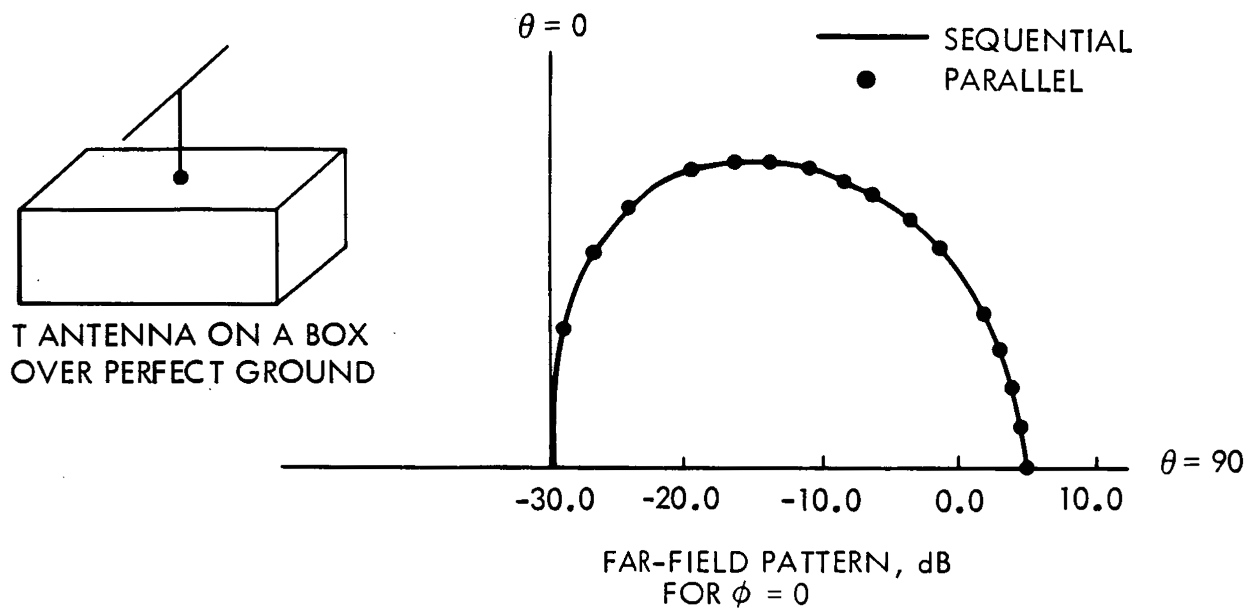


Figure 4.18 Radiation pattern of a T antenna on a box over perfect ground in $\phi=0$ plane.

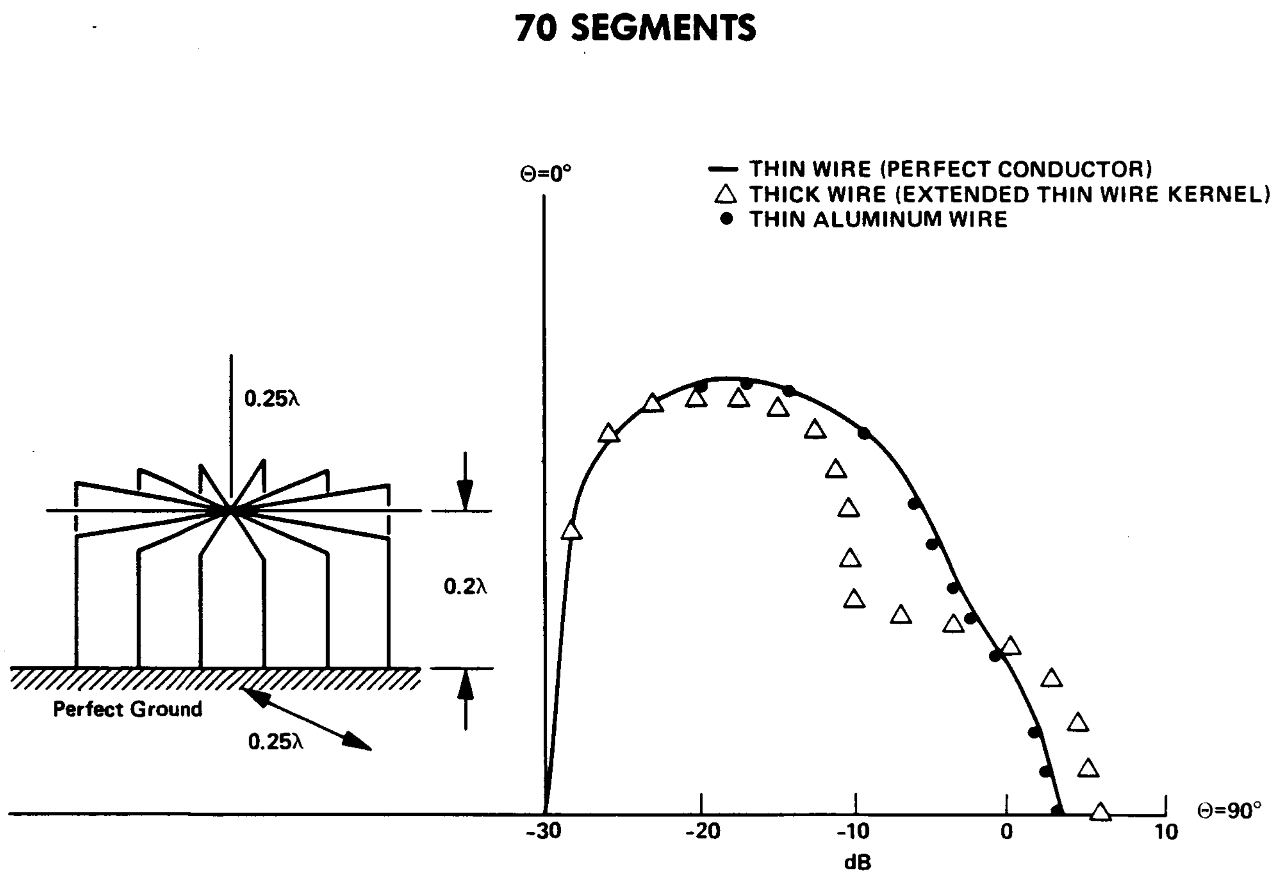


Figure 4.19 Radiation pattern of monopole on a pedestal over perfect ground in $\phi=0$ plane.

1. Timing

To analyze the timing performance of the parallel NEC program, two examples are considered:

a. Scattering by Sphere. The sphere of Section IV.D.2 is modeled by 120 surface patches without taking the symmetry into account. The interaction matrix for this problem is of size 240 * 240. The time to fill and factor this interaction matrix as well as the speedup factor are shown in the tables of Figures 4.20 and 4.21, respectively, as a function of the number of nodes used in the hypercube.

The times taken to do the fill and factor on one processor are 40.5 sec. and 3469 sec., respectively, while the same times are 25 sec. and 895 sec. for the VAX 11/750 computer. Despite the fact that the speed of the Mark III hypercube processors is almost equal to the VAX 11/750 processor, there is quite a large difference between the times shown above. For the fill time, the reason is that for each patch there are two equations and therefore two rows in the interaction matrix. In filling these rows sequentially there is some information that is computed once and then shared between the two rows. However, in the parallel code, since the two rows corresponding to each patch are in different processors, it is necessary to compute that information for every row in every processor which results in processing redundant information. For the factor time, the Householder transformation might be inherently slower than the LU decomposition method used for the sequential code resulting in a slower parallel code on one processor. Additional data comparing timing and the speed increase factor of the 32-node hypercube with the VAX is shown in Table 4.1.

The speedup factors for filling and factoring the interaction matrix are plotted in Figure 4.22. The speedup factor is defined as:

$$\text{Speedup factor} = \frac{\text{Time to run program on one processor}}{\text{Time to run program on Np processors}} \quad .$$

For this problem the top speedup factor (for 32 nodes) is 29 for filling and 23 for factoring of the interaction matrix.

It should be mentioned that usually the factor and fill time take more than 90% of the total time in the NEC program. Therefore the time taken to process the input, to solve the factored matrix, and to compute the far or near fields is insignificant compared to the total time.

b. Monopole on a Pedestal. The monopole on a pedestal of Section IV.D.1 is modeled by 20 ribs and a total of 290 segments here. Therefore, the interaction matrix is of the size 290 * 290. The fill and factor times vs. number of nodes are shown in the tables of Figures 4.23 and 4.24, respectively. Due to the fact that this problem does not fit on one

DIMENSION OF CUBE	NUMBER OF NODES	FILL TIME, sec	SPEEDUP FACTOR
0	1	40.5	1
1	2	20.5	1.98
2	4	10.3	3.93
3	8	5.2	7.79
4	16	2.6	15.58
5	32	1.4	28.93

Figure 4.20 Time and speedup factor for filling the interaction matrix of a sphere modeled by 120 surface patches vs. the dimension of the hypercube. Matrix size is 240 * 240.

DIMENSION OF CUBE	NUMBER OF NODES	FACTOR TIME, sec	SPEEDUP FACTOR
0	1	3469.0	1
1	2	1781.0	1.95
2	4	907.0	3.82
3	8	476.0	7.29
4	16	259.0	13.39
5	32	151.0	22.97

Figure 4.21 Time and speedup factor for factoring the interaction matrix of a sphere modeled by 120 patches vs. the dimension of the hypercube. Matrix size is 240 * 240.

Table 4.1 Timings for sphere

Number of Patches	Fill VAX sec	Fill 32-Node sec	Fill Increase	Factor VAX sec	Factor 32-Node sec	Factor Increase	Total VAX min	Total 32-Node min	Total Increase
80	10.6	0.6	17.7	260.5	50.4	5.2	4.5	1.0	4.5
120	25.4	1.4	18.1	895.2	148.4	6.0	15.3	2.5	6.1
168	52.5	2.6	20.1	2661.0	373.5	7.1	45.2	6.3	7.2
224	95.1	4.5	21.1	6236.0	836.6	7.5	105.5	14.0	7.5
288	-	7.5	-	-	1711.0	-	-	28.6	-
360	-	11.9	-	-	3254.0	-	-	54.4	-
440	-	17.7	-	-	5820.5	-	-	97.3	-

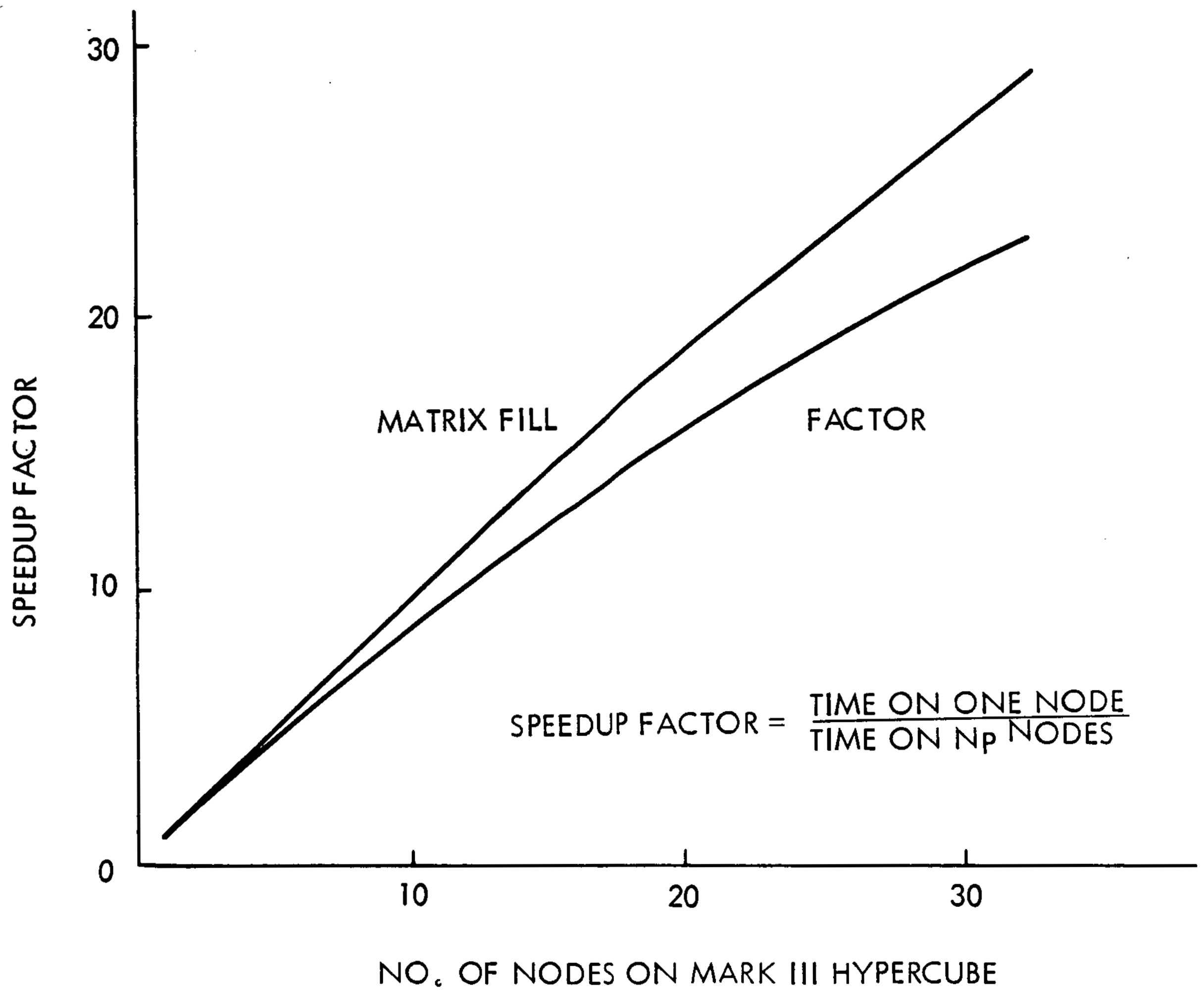


Figure 4.22 Speedup factor for filling and factoring the interaction matrix for scattering by sphere vs. number of nodes used. Matrix size is 290 * 290.

DIMENSION OF CUBE	NUMBER OF NODES	FILL TIME, sec		SPEEDUP FACTOR	
		SOURCE-LOOP SEQUENTIAL	SOURCE-LOOP PARALLEL	SOURCE-LOOP SEQUENTIAL	SOURCE-LOOP PARALLEL
0	1	1725.0	1725.0	1	1
1	2	876.0	915.3	1.97	1.88
2	4	445.5	506.6	3.87	3.4
3	8	230.3	292.6	7.49	5.9
4	16	122.6	158.7	14.07	10.9
5	32	68.8	81.5	25.07	21.2

Figure 4.23 Time and speedup factor for filling the interaction matrix of the monopole on a pedestal vs. the dimension of the hypercube. Results from source-loop parallel and source-loop sequential codes are shown. Matrix size is 290 * 290.

DIMENSION OF CUBE	NUMBER OF NODES	FACTOR TIME, sec	SPEEDUP FACTOR
0	1	6110.0	1
1	2	3141.0	1.95
2	4	1615.0	3.78
3	8	841.0	7.27
4	16	454.0	13.46
5	32	259.0	23.59

Figure 4.24 Factor time and speedup factor for interaction matrix of the monopole on pedestal vs. the dimension of the cube. Matrix size is 290 * 290.

node, the fill and factor times are computed by extrapolation of the results from a higher number of nodes.

For the fill time the two parallel codes discussed in Section IV.C.2 are used. It is evident that for this problem, the source-loop sequential code is faster than the source-loop parallel code for any number of nodes used. This is due to the fact that, for this example, processing of the information in the source loop takes less time than it takes to communicate that information to other nodes. The fill and factor time for the VAX 11/750 computer are 1830 sec. and 1693 sec., respectively. The fill time is very close to the time on one node due to the fact that the parallel program for wires on one node is almost identical to the VAX sequential code. However, the factor time is quite different for the same reason as discussed in Section IV.D.1. Additional data for timing and speed increase factors for the 32-node Mark III Hypercube to those for the VAX 11/750 are shown in Table 4.2.

The speedup factor for filling and factoring the interaction matrix vs. the number of nodes used in the hypercube is shown in Figure 4.25. These curves are almost linear; however, the SLSC shows a better speedup factor across the range for the number of nodes than the SLPC.

2. Fixed Problem

When analyzing how a particular algorithm scales for different hypercube sizes, we need to keep the size of the problem fixed within a node. If the total problem size remains the same, the amount of work assigned to each processor decreases as the number of processors in use increases. Then, as a result, a significant portion of the time is spent idly waiting for other processors to complete or communicate rather than performing useful work.

To demonstrate how the FILL algorithm of parallel NEC scaled with the size of the hypercube, the input data size was determined so that for each run all processors received 5,000 elements, that is

$$\frac{n^2}{\text{No. Processors}} = 5000$$

As the number of elements in a row increases, the number of rows for which a processor is responsible decreases. Figure 4.26 shows the results for the runs on the hypercube beginning with a dimension of 0 (1 node) and proceeding up to a dimension of 5 (32 nodes). As the number of processors increases, the parallel algorithm demonstrates excellent scalability. The slight increase in time required for larger size hypercubes is expected. The number of elements per node is the product of N_T (the number of elements in a row of the interaction matrix) and the number of rows in that node. During the fill there is an initial computation which is performed N_T times. As the number of rows per node decreases, the size of N_T increases and, as such, slightly affects the overall time.

Table 4.2 Timings for monopole on a pedestal problem.

Number of Wires	Fill VAX sec	Fill 32-Node sec	Fill Increase	Factor VAX sec	Factor 32-Node sec	Factor Increase	Total VAX min	Total 32-Node min	Total Increase
210	943.4	33.9	27.8	586.2	110.4	5.3	25.5	2.4	10.6
230	1125.5	41.3	27.3	817.6	139.2	5.8	32.4	3.0	10.8
250	1307.1	44.8	29.2	1078.9	173.6	6.2	39.8	3.6	10.9
270	1541.8	53.2	29.0	1423.6	214.4	6.6	49.4	4.5	11.1
290	1807.6	62.5	28.3	1771.1	259.7	6.8	59.0	5.4	11.0

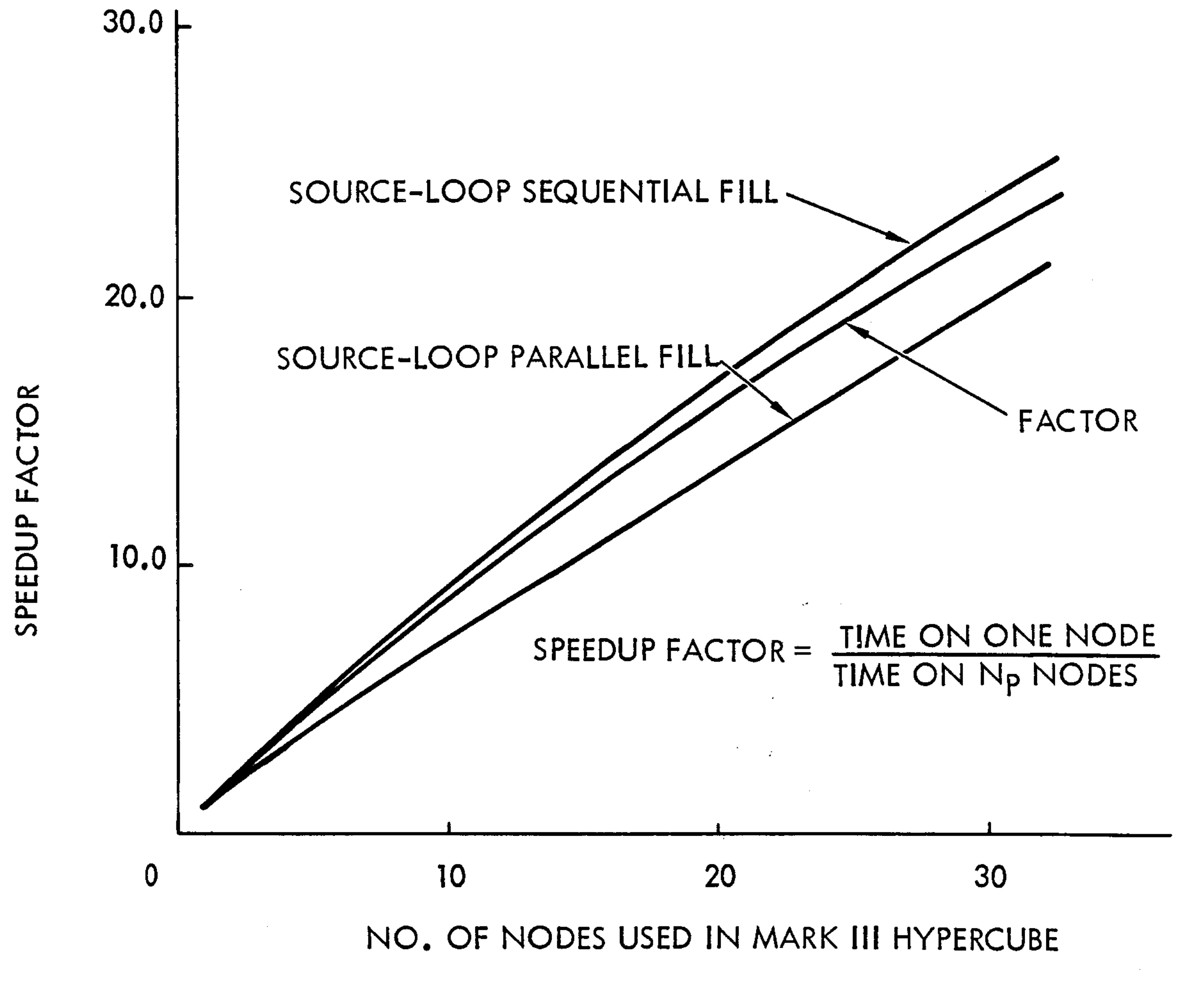


Figure 4.25 Speedup factor for fill and factoring the interaction matrix of the monopole on a pedestal vs. number of nodes used. Results from the source-loop parallel and source-loop sequential codes are shown for matrix fill. Matrix size is 290 * 290.

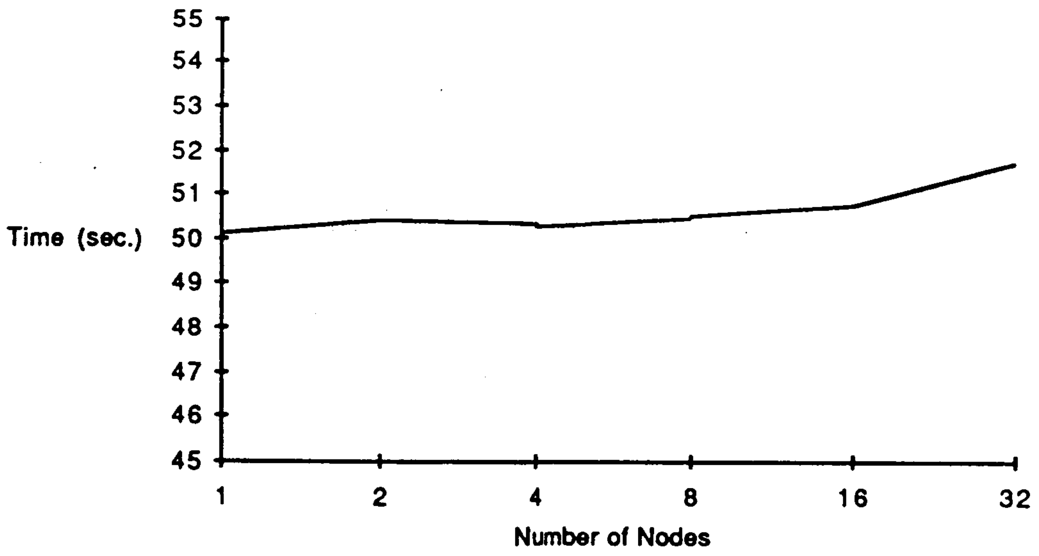


Figure 4.26 Scaling of the performance of the parallel fill algorithm where the problem size per node remains fixed at 5000 elements.

It is more difficult to analyze how the factor algorithm of the parallel NEC code scales with the size of the hypercube. One could construct a fixed case data set based on the fact that factorization algorithms are of order n. However, as the number of elements in the overall matrix increases for larger hypercubes, so does the number of transformations required to perform the factorization. Each transformation is preceded by the BROADCAST of the pivotal column as well as some ancillary data. The results from different sizes of hypercubes would be a comparison of problems with considerably different computation and communication requirements.

3. Analysis of Performance

In using the two parallel codes, it is observed that for the 290 segment monopole on a pedestal, the source-loop sequential code (SLSC) consistently achieved lower fill time than the source-loop parallel code (SLPC). Even though better time was obtained from SLSC for most other examples that were tried throughout this study, there are some special cases where SLPC yields lower fill time. To understand this phenomena better, we should study the source loop in more detail. As was discussed in Section IV.C.2 the source segment information is computed inside a DO-loop with index variable J, where J goes from 1 to N (the total number of segments). For problems where the structure is modeled by wires, the time to process this information can be substantial and therefore the SLPC was developed to communicate the data, instead of computing it sequentially in every element, as in the SLSC. For wire problems, each wire segment can be connected to several other segments as shown in Figure 4.27. In general each segment can be connected to nseg number of other segments. Since the basis function on each segment is extended over the adjacent segments, the number of local boundary equations to be solved for eliminating two out of three coefficients in each basis function increases as the number of connected segments increases. In most problems wire junctions have only 2 wires connected to them (simple junction) and therefore source-loop information can be processed very fast without any need to use communication between nodes (SLSC should be used). However, if a problem is modeled with multi-wire junctions one has to look at the ratio of the number of these junctions to the number of simple junctions as well as the number of nodes used in the hypercube to select either SLSC or SLPC. Note with a higher number of nodes, more communication is done. To understand this better, the monopole on pedestal example is modeled with a different number of wire segments but with 20 ribs in modeling of the pedestal. Therefore, only one multi-wire junction with 21 segments is always present and by changing the number of segments, only the number of simple junctions changes. These models were run on a 32-node hypercube, for maximum communication effect, and the fill time obtained from SLSC and SLPC are plotted vs. the total number of segments in the model as shown in Figure 4.28. It can be seen that for a number of segments less than 90, the SLPC is faster which is due to a relatively small number of simple junctions. As the number of segments is increased above 90, the number of simple junctions is increased and since, for these junctions, computing the source information is faster than communicating it, the SLSC yields better fill time.

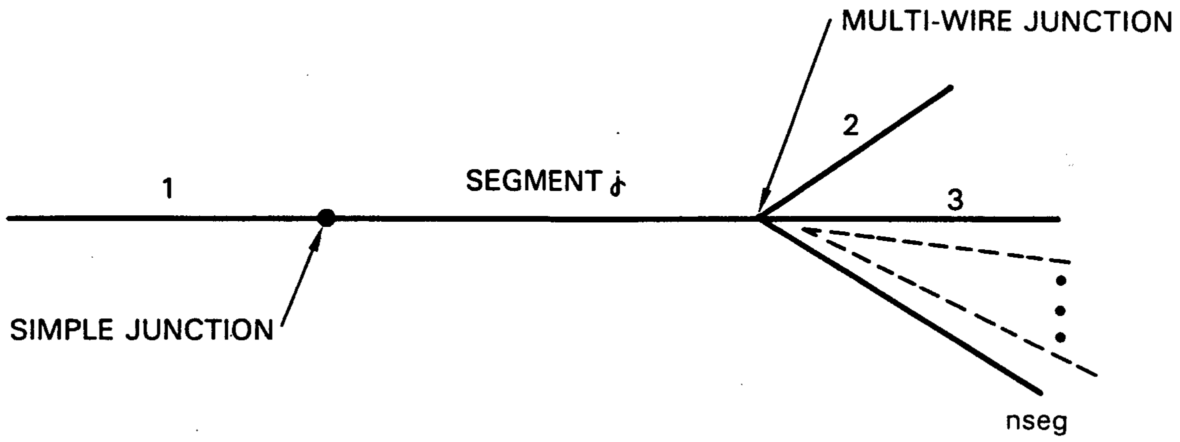


Figure 4.27 General wire segment connection.

C-2

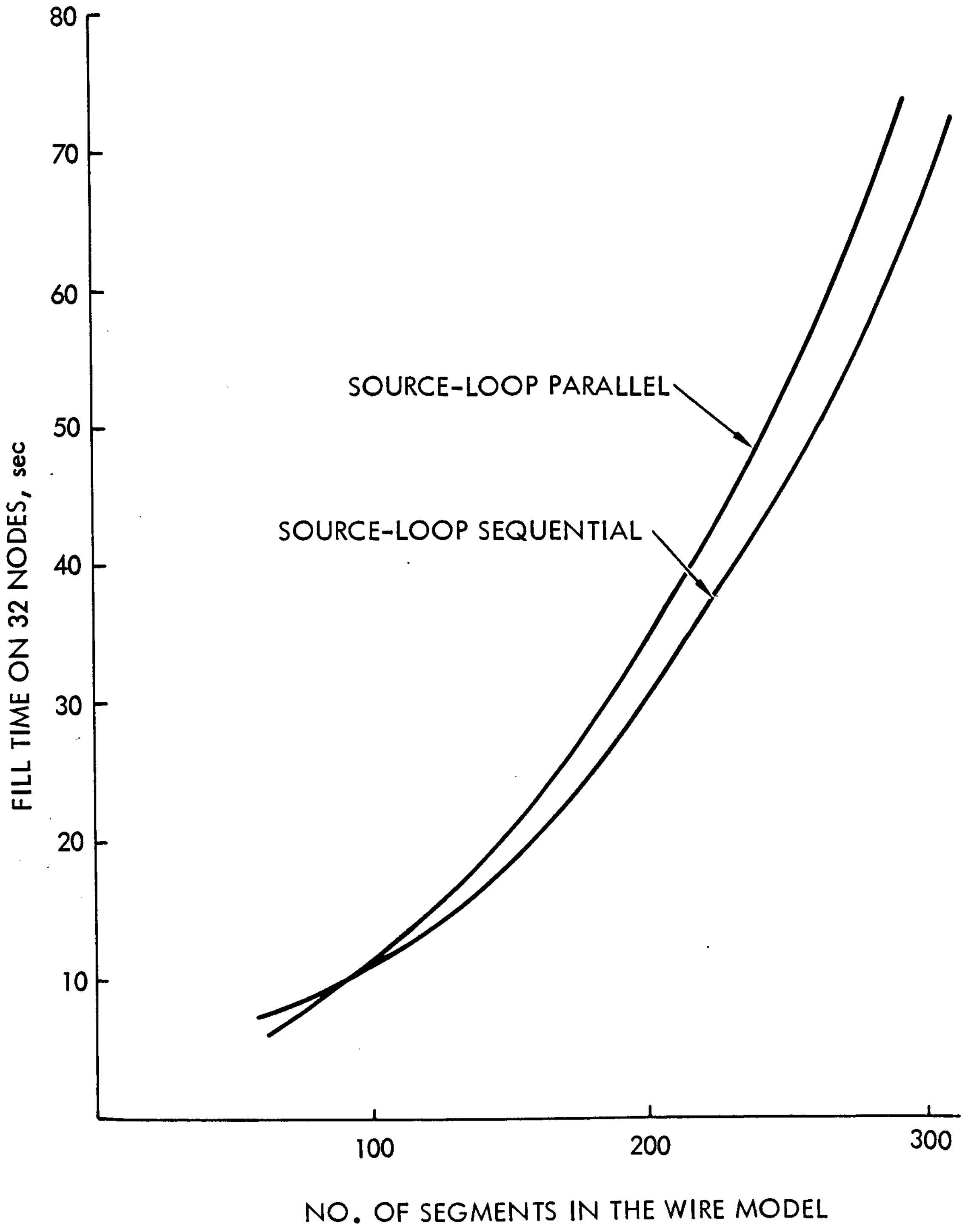


Figure 4.28 Time for interaction matrix fill for monopole on a pedestal vs. the number of segments in the model for the source-loop parallel and source-loop sequential codes. 32 nodes used.

F. OPTIMIZATION

The JPL/Caltech Mark III Hypercube project is a research and development effort in developing parallel computing systems. Along with the hardware development is the corresponding software effort to create operating systems and software support tools to aid the applications programmer. Two operating systems have been developed. One uses synchronous message passing requiring two nodes to be synchronized at the time of communication. It is called the Crystalline Operating System (CrOS). The other is Mercury which allows asynchronous communication utilizing message buffers for queueing up incoming and outgoing messages. At this point in the software development CrOS is supported for both C and FORTRAN, however Mercury does not yet extend to FORTRAN or hybrid codes. One possible future optimization for the parallel NEC code is the addition of some Mercury communication capabilities to such code components as the back substitution for solution of the equation.

Further optimizations of communication calls are also being developed. For instance, the COMBINE function which combines data from all nodes using some programmer-specified function such as an exclusive OR currently interrupts the application processor in the course of the instruction. This instruction is also limited to manipulation of no more than 512 bytes of data. Operating system software optimization is being made to locate this instruction solely in the communication processor and to have it operate on an arbitrary number of bytes.

Hardware developments for the Mark III Hypercube continue as well. Currently the hypercube uses the Motorola 68881 floating point processor. A daughterboard addition is being implemented to house the powerful Weitek floating point accelerator chip set. The current floating point processor delivers 60-120 KFlops per node. Preliminary tests of the new daughterboard demonstrate up to 10 MFlops per node of performance. These daughterboards will be added to the existing Mark III nodes during the fall of 1987.

Additional optimizations to the current parallel NEC code can be considered: for instance, the identification of the most time consuming FORTRAN portions of the parallel code and conversion of these segments to the C programming language. When using pointers to access arrays and inserting frequently-used variables into registers, C has been demonstrated to achieve a speedup over the analogous code written in FORTRAN. In the current version, only the matrix inversion and the complex arithmetic routines are in C.

Another fairly minor change can be made to the parallel NEC code which would increase its efficiency somewhat. At present, the calculation of the current vector I from the basis function amplitude vector F is being done in the CP in the original sequential subroutine CABC. The part of the subroutine which calculates the current in the patches is not easily done in parallel because it involves taking the two tangential patch current components and relating them to three x, y and z components, thus changing the size of the solution vector. The part of this subroutine which calculates the current in the wires can easily be done in parallel in the hypercube. Thus, the part of the subroutine dealing with the patches could be separated and left in the CP code, and the part dealing with wires moved to the hypercube code. This would (1) decrease the time for this calculation and (2) lessen the chance that, for multiple E cases, the hypercube would have to sit idle while the CP performs the current and radiation calculations (see Figure 4.4).

REFERENCES

[4-1] Burke, G. J. and Poggio, A. J., "Numerical Electromagnetics Code (NEC) - Method of Moments," (Lawrence Livermore National Laboratory, 1981).

[4-2] Bierman, G. J., "Factorization Methods for Discrete Sequential Estimation--Mathematics in Science and Engineering Volume 128," (Academic Press, 1977).

SECTION V

TEST CASE: SCATTERING FROM A CONDUCTING SPHERE

In this section, results from both the parallel NEC and PFDTD codes are presented for the same sponsor-selected test case: scattering from a perfectly conducting sphere. A comparison is made of the results for both near fields and radar cross sections. Results are also compared to the exact (analytic) solution.

A. DESCRIPTION OF THE TEST CASE

The radius of the perfectly conducting test sphere is chosen to be a=47.714651 m. The incident wave is a linear polarized plane wave with wave vector k in the -x direction, an electric field of 1 V/m in the -z direction, and a magnetic field of 2.654e-03 A/m in the -y direction (see Figure 5.1). The sphere is in a vacuum. Near electric and magnetic fields are compared for ka=3.0 along a line in a plane in front of the sphere, behind the sphere and below the sphere. The radar cross sections are compared for ka values from 1 to 4.5 in steps of 0.5.

1. Test Case for Parallel FDTD

For the test cases with ka=1.0-3.5 PFDTD uses a 74 x 74 x 74 grid with a cell size of 2.6508 x 2.6508 x 2.6508 m. The total domain is 196 x 196 x 196 m. For ka=3.0 and the above test sphere, the wavelength is 99.93 m. PFDTD uses rectangular geometry only and thus the conducting sphere is modeled as a collection of perfectly conducting cubes, leading to a "staircase" profile. Since the grid spacing is 2.6508 m, there is on the order of a 5% variation in the "sphere" radius. To obtain the radar cross sections, the near fields are integrated on the six planes located at x, y and z of ± 76.8736.

For the test case with ka=4.0 and 4.5, PFDTD uses a grid spacing of 2.3857 m to obtain better resolution. The test cases with ka=2.0 and 3.5 were also calculated with this smaller cell size spacing to obtain more accurate results on the radar cross sections.

2. Test Case for Parallel NEC

For the test cases with ka=1.0-3.0, parallel NEC modelled the sphere with a total of 80 surface patches. Figure 5.2 illustrates the patch modelling of 1/8 of the sphere surface. This pattern was repeated over the remaining portions of the sphere surface. These patches were constructed in a manner analogous to that used in Example 9 of Section IV of the NEC manual [Ref. 5-1]. The code's symmetry option was NOT invoked for these test cases so the matrix equation to be solved was of order N_T=160. For the NEC input, the incident wave specified above corresponds to a plane wave with θ=90 degrees, ϕ=η=0 degrees.

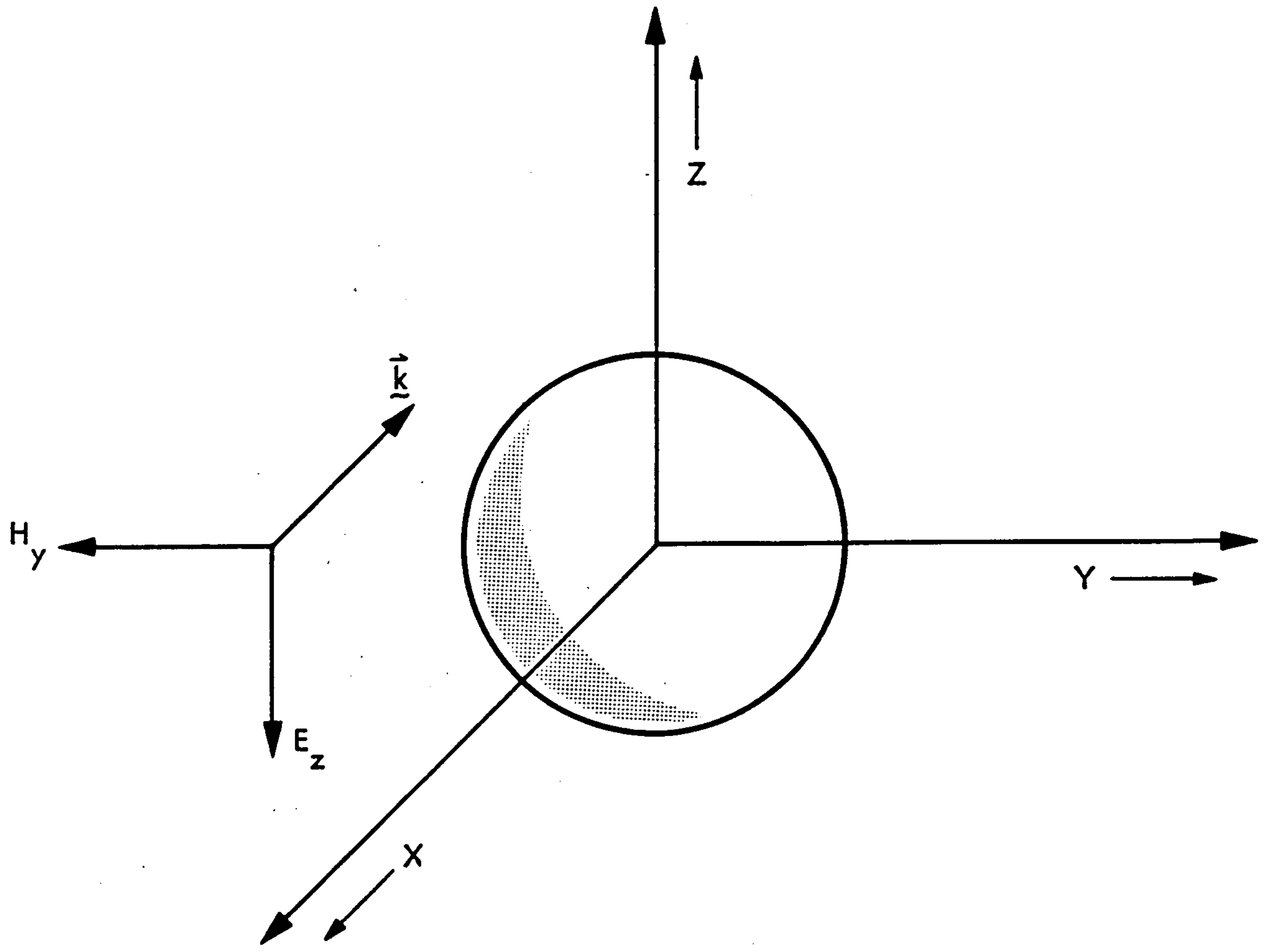


Figure 5.1 Geometry of the test case: scattering from a perfectly conducting sphere. The sphere is centered at the origin. A plane wave is incident from the -x direction.

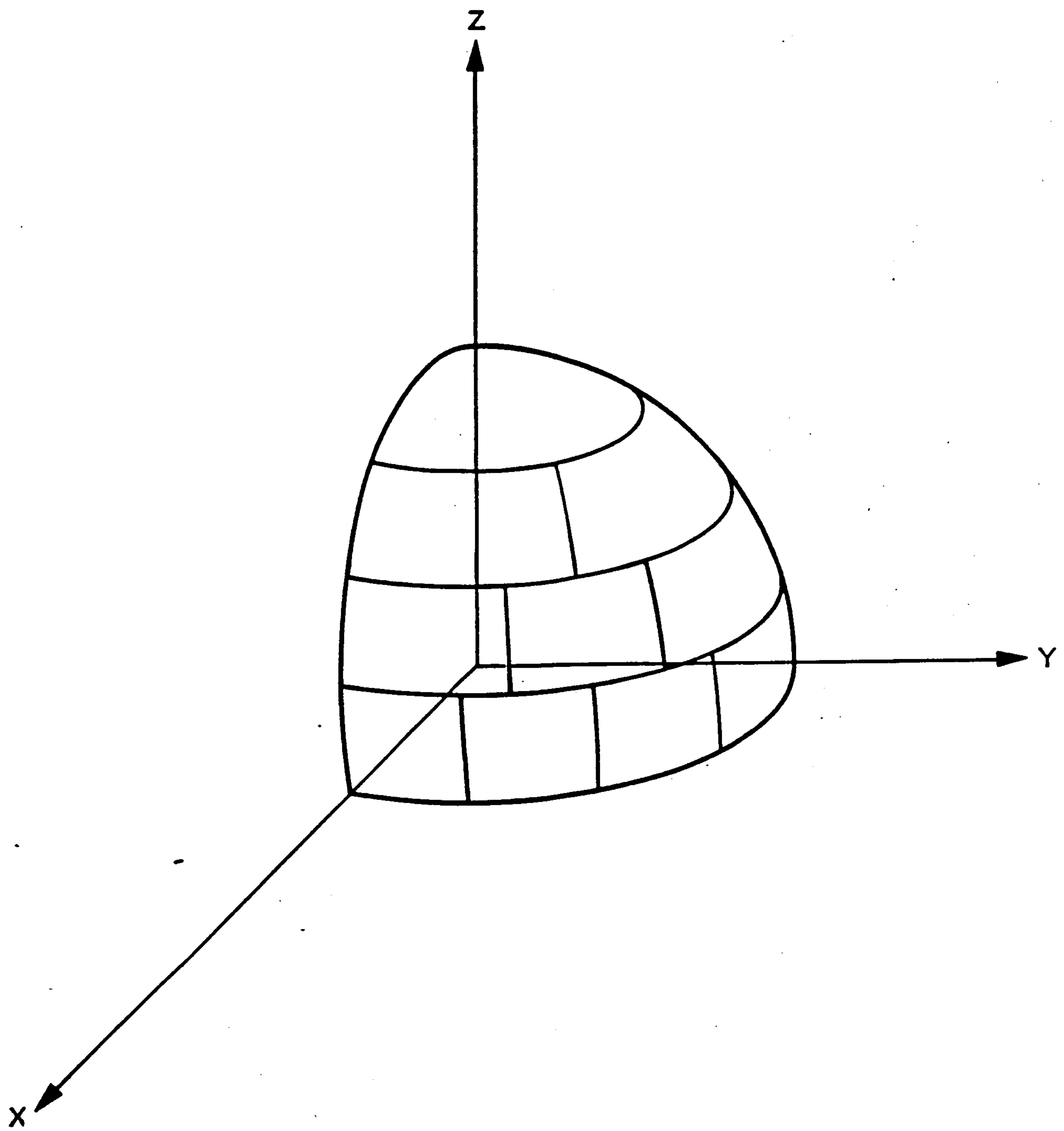


Figure 5.2 Patch model of a sphere for the 80 patch model used in the parallel NEC code. The pattern was repeated over the remaining 7 sections of the sphere surface.

For the test cases with ka=3.5-4.5, the sphere was modeled with 168 surface patches in order to obtain sufficient resolutions. For ka=5.0, 288 patches were used and for ka=6.0 and 6.5, 360 patches were used. The case ka=3.0 was done with 80, 120, and 168. It should be noted that test cases run on the parallel NEC code always agreed with the sequential VAX NEC code results to all significant figures. The times for the runs for 80, 120, and 168 patches can be found in Table 4.1 in Section IV.

3. Exact Solution

The exact solution for this test case is attributed to D. Mie. The solution uses spherical wave functions and boundary value techniques to find the terms of the resultant Mie series for the scattered field. The same formulation is used regardless of the composition of the sphere, e.g., its dielectric constant. Only the coefficients of the terms change. A detailed mathematical discussion of the solution can be found in Ruck, et al. [5-2] or Bowman, et al. [5-3].

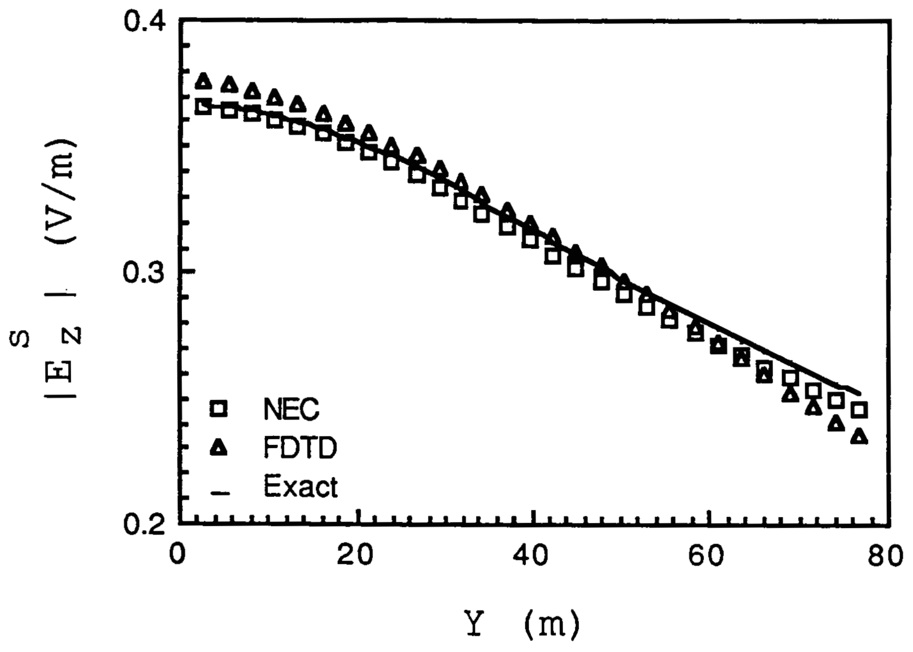
A FORTRAN code for the evaluation of the series was obtained from Dr. Sembiam Rengarajan, California State University, Northridge. The code was used to generate the scattered near fields only; far field results were not generated. The code was tested by comparison with near field plots in [5-3].

The Mie series is an infinite sum and, thus, in practice it must be terminated after a finite number of terms. For a sphere with ka on the order of 3, the series can be truncated at ka +10 terms with an error in the near fields of no more than 5% [5-2].

B. COMPARISON OF RESULTS

1. Front Plane Comparison of Near Fields

Results from both parallel codes and from the analytic solution for the magnitude and phase of the scattered field Ez in a plane in front of the sphere are shown in Figure 5.3. The comparison is made at x=76.8736, z=22.5319, for 29 values of y starting at y=2.6508 and incremented in steps of 2.6508 (the grid spacing for PFDTD). The magnetic field Hy is shown in Figure 5.4 for the same positions. Because of the polarization for the incident wave, these field components are much larger than the others in this plane. It can be seen that there is excellent agreement between the results of both codes and the analytic solution. The maximum difference between the parallel NEC code and the analytic solution for the amplitude of Ez is about 2.5% and the maximum difference between PFDTD and the analytic solutions is about 6%. Over most of the range, it can be seen that the error is generally much smaller.



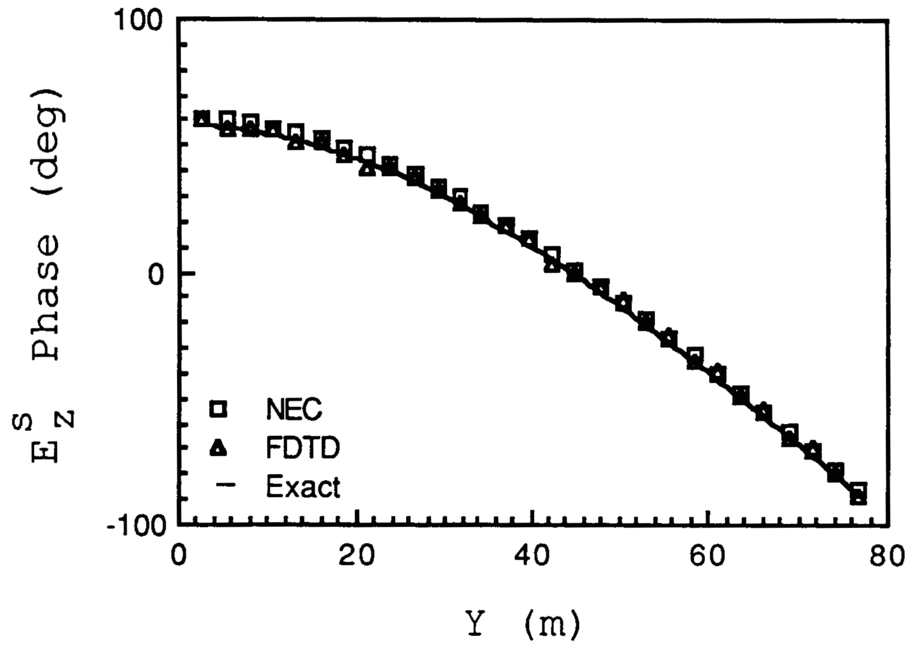
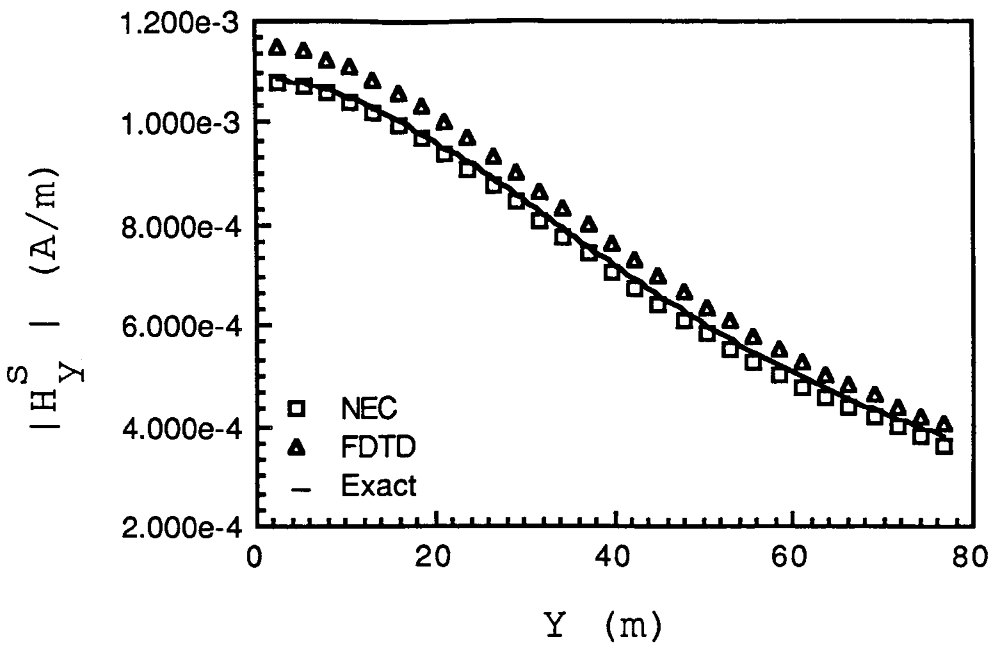


Figure 5.3 Comparison of NEC, PFDTD, and exact results for the amplitude and phase of the scattered Ez field. The comparison here is in front of the sphere, x=76.8736, z=22.5319.



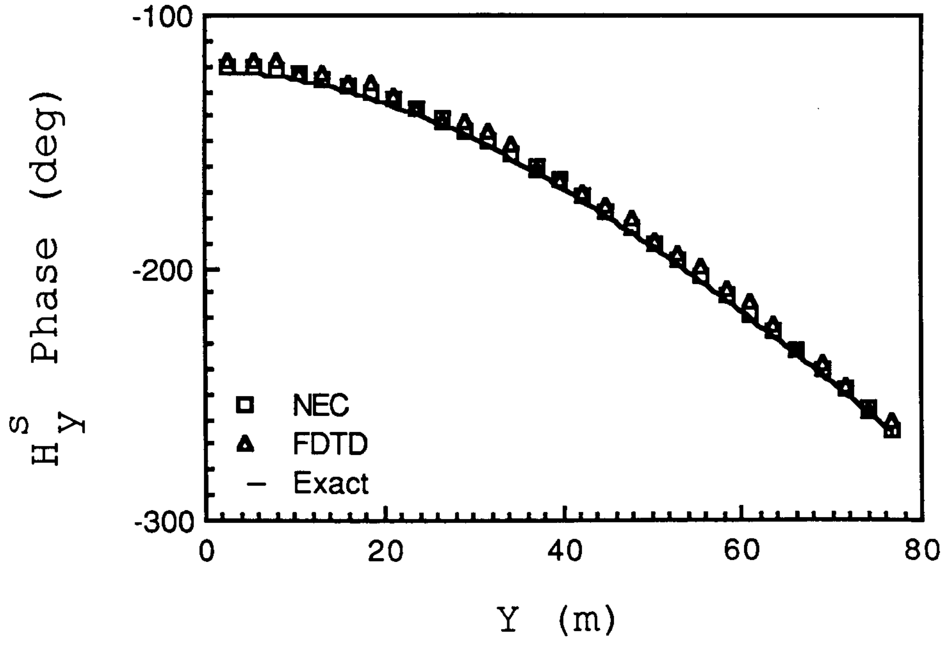


Figure 5.4 Comparison of NEC, PFDTD, and exact results for the amplitude and phase of the scattered Hy field. The comparison here is in front of the sphere at the same positions as in Figure 5.3.

2. Back Plane Comparison of Near Fields

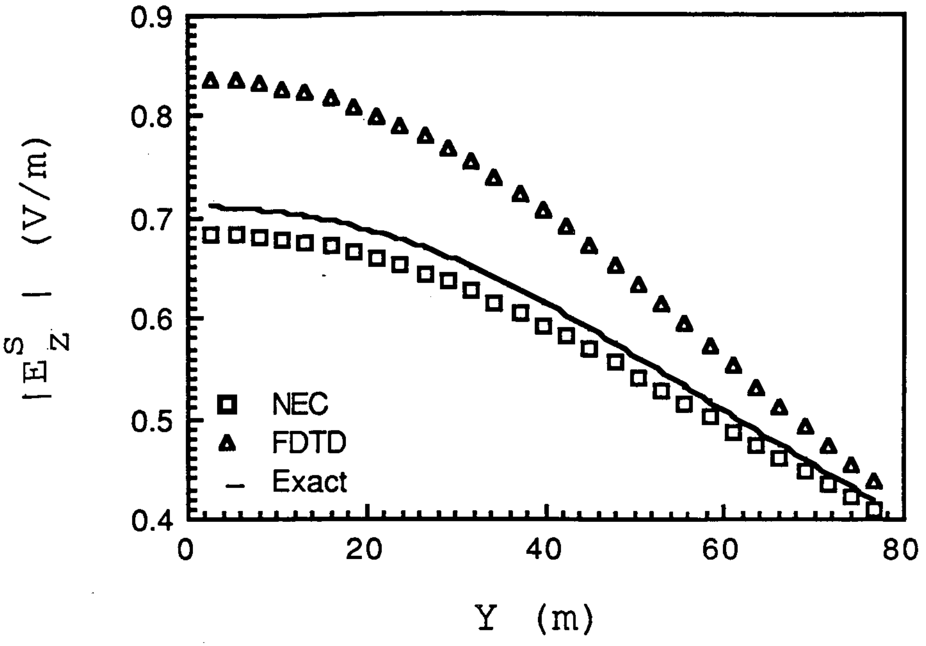
Results from both parallel codes and the analytic solution for the magnitude and phase of the scattered field Ez in a plane in back of the sphere are shown in Figure 5.5. The comparison is made at x=-76.8736, z=22.5319, for 29 values of y starting at y=2.6508 and incremented in steps of 2.6508 as in the front plane comparison. The magnetic field Hy is shown in Figure 5.6 for the same positions. Here, the agreement between the exact and NEC results is still quite good, although the percentage difference is as much as 4% for small y values. For this plane, the PFDTD results differ from the NEC and exact results by about 20%. The difference drops to under 10% in the large y region. The smaller y region falls in the "shadow" of the test sphere. The discrepancy between the PFDTD code and the exact solution is discussed in Section V.C.

3. Bottom Plane Comparison of Near Fields

Results from both codes for the magnitude and phase of the scattered field Ey in a plane below the sphere are shown in Figure 5.7. The comparison is made at x=-23.8573, z=-76.8736 for 29 values of y starting at y=1.3254 and incremented in steps of 2.6508. The scattered magnetic field Hy is shown in Figure 5.8 for the same positions. This region is not in the shadow and, except for a few anomalous points, the agreement between the two codes is good. The magnitude and phase for the scattered Ey fields differ by no more than 14% except for the first two anomalous points. The magnitude and phase of the Hx scattered fields differ by no more than 14% except for the first three points.

The explanation for the anomalous behavior lies in the fact that the absolute values for the fields at these points are smaller than the incident field amplitudes by about two orders of magnitude. The PFDTD code has difficulty in computing these small field values accurately. The anomalous phase values result from the method used by PFDTD to calculate the phases: At each time step, PFDTD has the amplitude and an approximation to the first derivative of the amplitude of the discrete field components one time step back. PFDTD, then, approximates the first derivative of the amplitude for the current time step. If a change occurred in the first derivative between the previous and current time step, PFDTD assumes a peak or valley has occurred in the amplitude. The code uses a second derivative test to distinguish between a peak and valley. If a peak occurs, PFDTD records the current iteration count. PFDTD subtracts a reference iteration number from the iteration number. From the known time increment dt, wavenumber k, and the speed of light c, PFDTD calculates a phase relative to the reference time step in radians.

There are two results of this phase calculation. Because of the finite time increment dt, the phase plots have staircase profiles. If the amplitudes of the fields at selected test points are small compared to the magnitude of the incident fields, the waveform may not be sinusoidal. Recorded time step iterations may not be reliable for these cases.



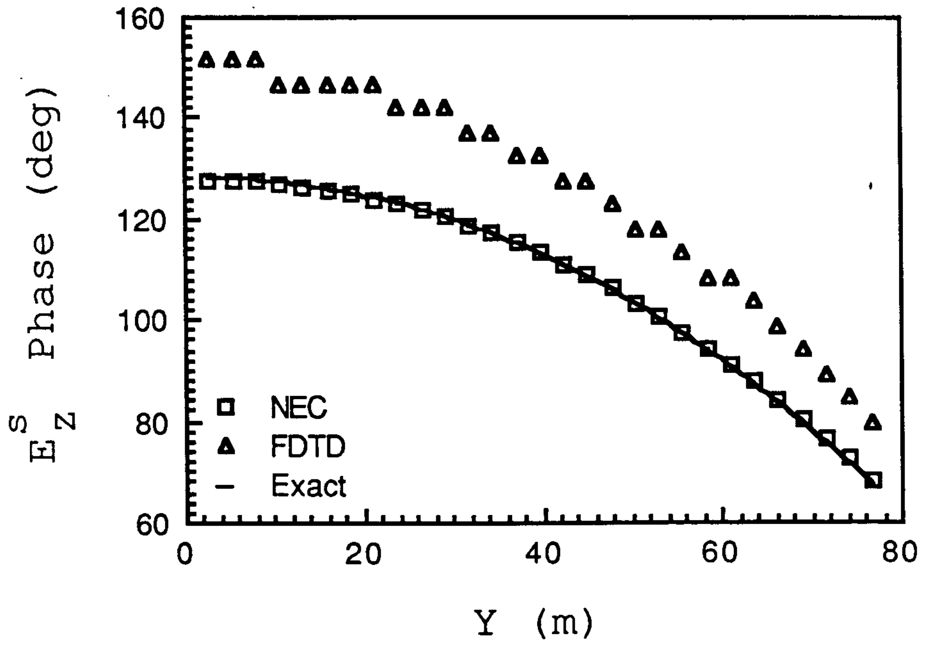
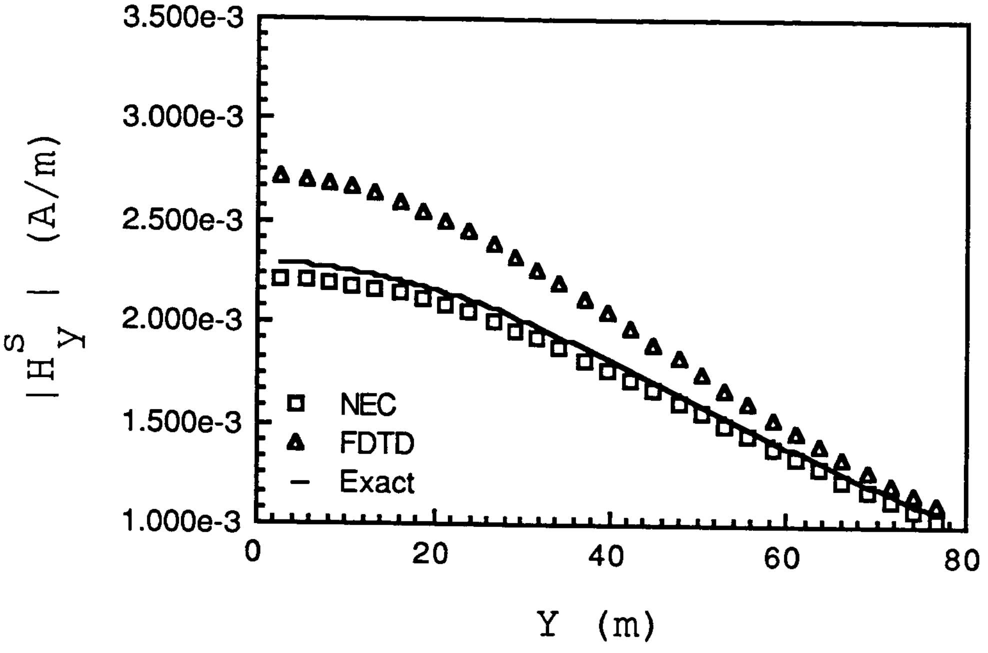


Figure 5.5 Comparison of NEC, PFDTD, and exact results for the amplitude and phase of the scattered Ez field. The comparison here is behind the sphere, x=-76.8736, z=22.5319.



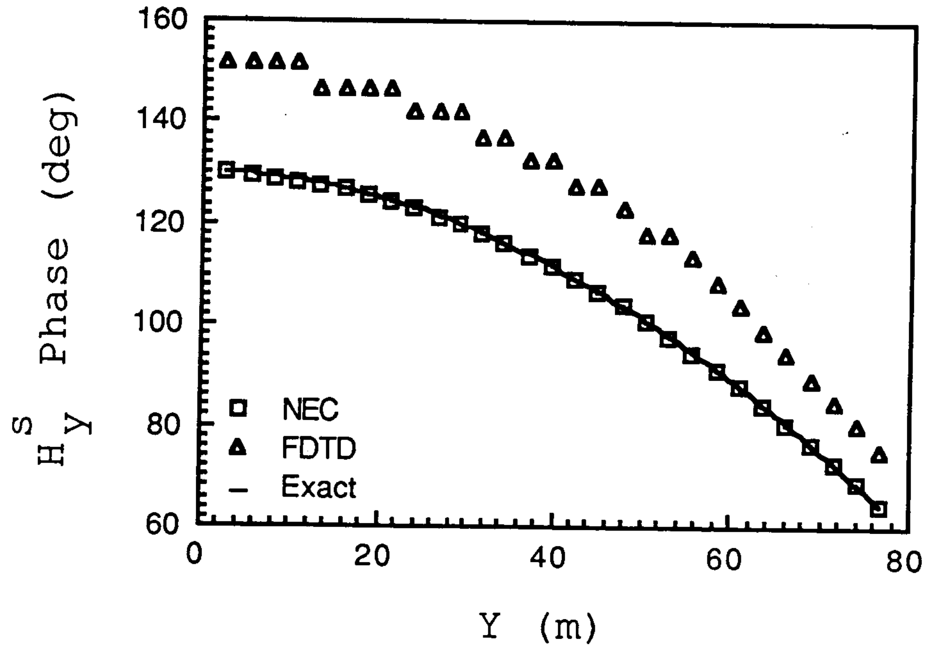
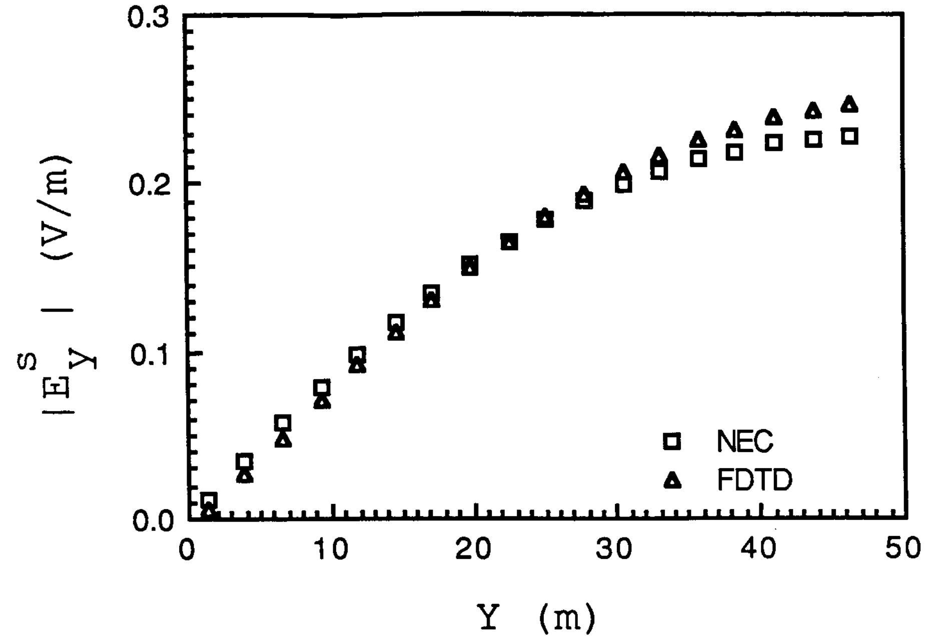


Figure 5.6 Comparison of NEC, PFDTD, and exact results for the amplitude and phase of the scattered Hy field. The comparison here is behind the sphere at the same positions as in Figure 5.5.



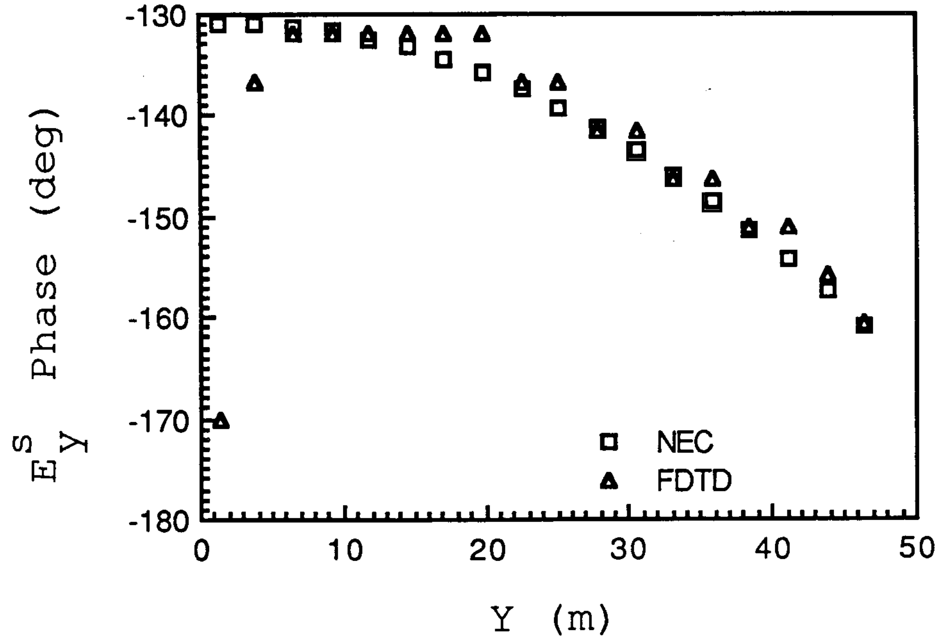
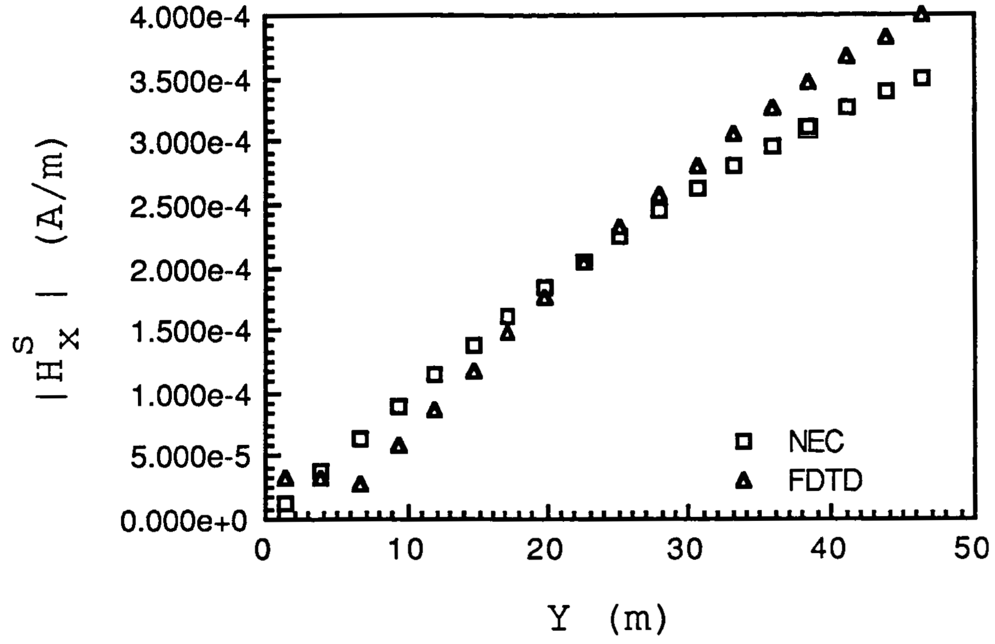


Figure 5.7 Comparison of NEC, PFDTD, and exact results for the amplitude and phase of the scattered Ey field. The comparison here is below the sphere, x=-23.8573, z=-76.8736.



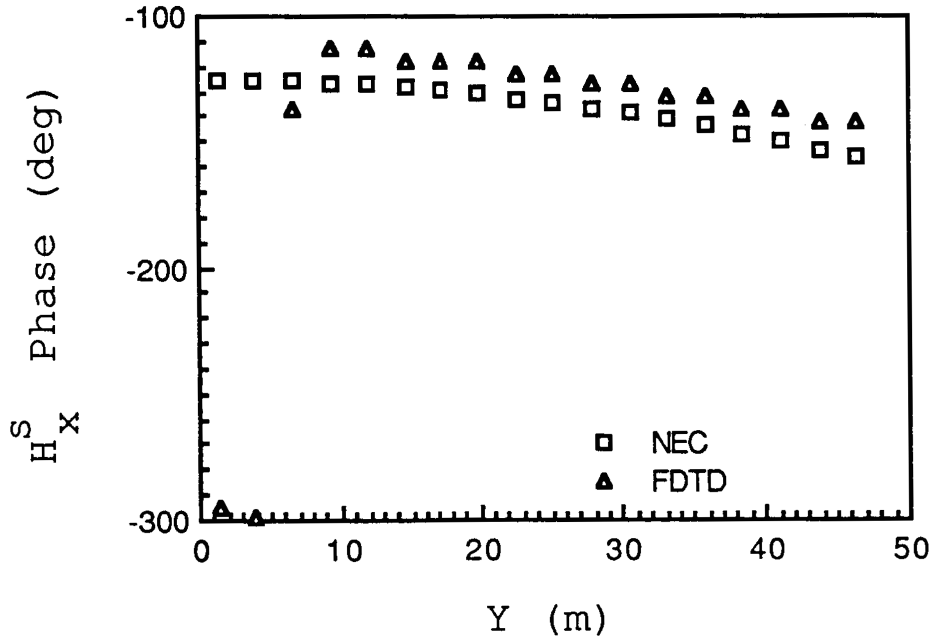


Figure 5.8 Comparison of NEC, PFDTD, and exact results for the amplitude and phase of the scattered Hx field. The comparison here is below the sphere at the same positions as in Figure 5.5.

4. Comparison of Radar Cross Sections

Radar cross sections from the two codes are compared to the exact solution in Figure 5.9. The exact solution is based on a figure taken from [5-2]. The definition of the radar cross section is in Part II of the NEC manual [5-1] in the description of the subrouting RDPAT which calculates the far field quantities. Parallel NEC results for the test case are shown as dots. PFTD results for a cell size of 2.6508 m are shown as squares and results for a cell size of 2.3857 are shown as triangles.

The parallel NEC results plotted in Figure 5.9 are in very good agreement with the exact results with the exception of the results for ka=2.5 and 4.5 which are off by about 20%. The discrepancy at ka=2.5 is due to large internal fields which are excited inside the sphere in the region around a cavity resonance at ka=2.744, as discussed in Example 9 in Part III of the NEC manual [5-1]. A different cavity resonance is the most likely cause of the error at ka=4.5 as well. This pitfall of the NEC approach is discussed in Sec. V.C. below.

The PFDTD code results are in fairly good agreement. Considering only the points with the finer resolutions, the PFDTD results for ka=4 and under are within 20% of the exact result. The case ka=4.5 is off by 50%. At ka=4.5, the wavelength is 67 m, or about 3 times the grid spacing. Apparently this case is simply beyond the resolution of the code. This problem was the largest size possible to run in the 32-node Mark III Hypercube. More nodes would be needed to resolve this and higher ka cases.

C. DISCUSSION OF RESULTS

From the plots shown above (Section V.B.2), one can conclude that in general PFDTD is less accurate than NEC for scattering from a perfectly conducting sphere. NEC's method of moments approach is known to be very accurate for such steady-state scattering problems. For the problem of plane wave scattering from a perfectly conducting cube, Taflove (see [3-3] of Section III.A) claims to obtain near fields accurate to within ±3% when using a unit cell of size $\delta = \lambda/20$. Clearly, then, FDTD should be capable of accurate results.

1. Sources of Error in the NEC Code

The NEC code is known to give erroneous answers when large internal fields are excited in the object modeled. For a conducting sphere, a resonant TM-101 cavity mode exists at ka=2.744 [5-1]. However, the patch model of a sphere used in the test case is not a perfect sphere since NEC only utilizes the center and total area of the patches in the model and not their shape. Apparently, this results in giving significant width to the ka=2.744 cavity resonance. In Figure 5.10, the radar cross section in the region around this resonance is plotted. NEC results for the test case (plane wave incident in the -x direction) are plotted as filled dots. Figure 5.9 showed only the results at ka=2.5 and ka=3.0. Here, results at ka=2.5, 2.6, 2.7, 2.75, and

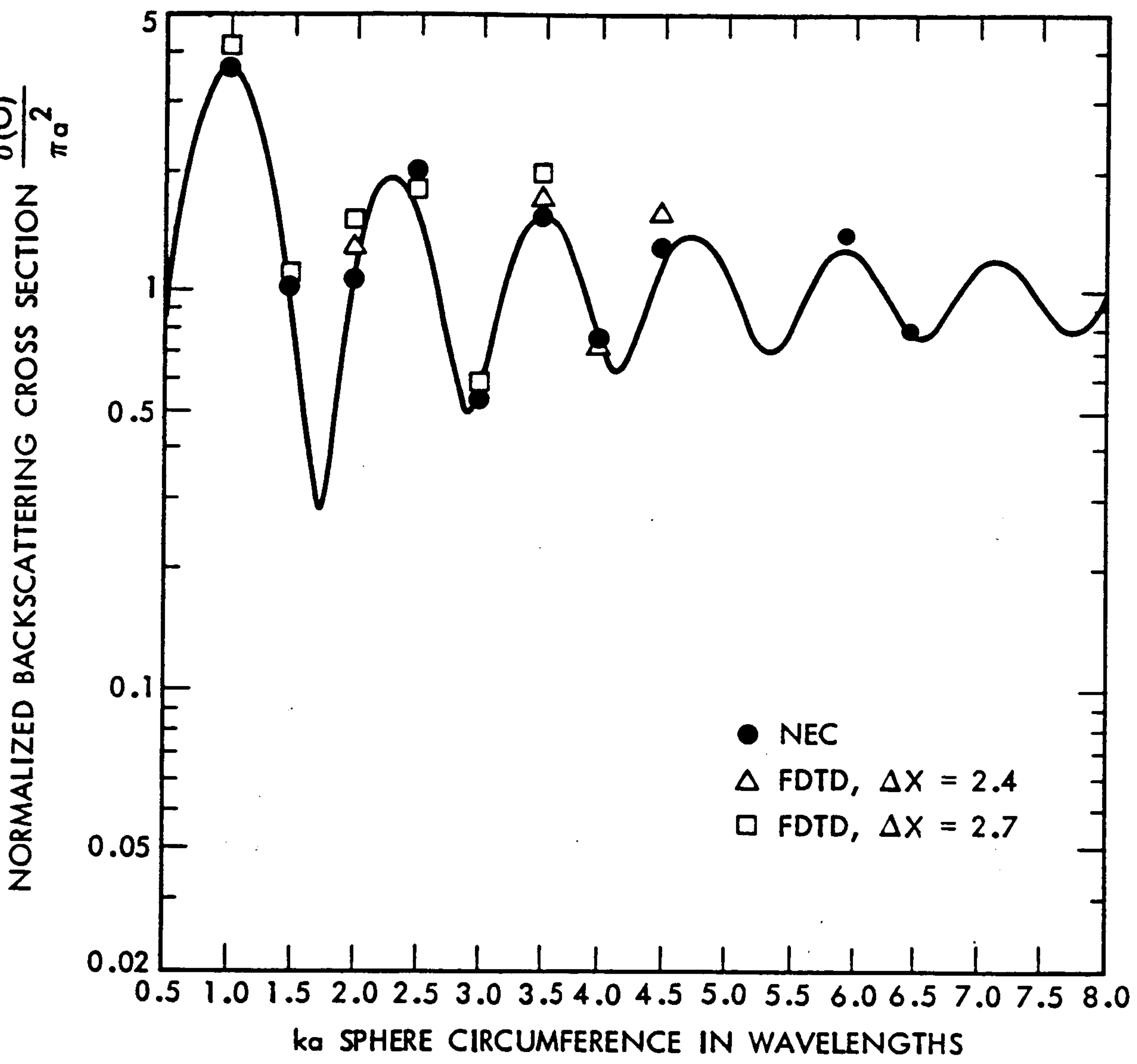


Figure 5.9 Comparison of parallel NEC and PFDTD results with exact radar cross sections. The curve is the exact results, the dots are from NEC and the squares and triangles are from the PFDTD code for two different cell sizes.

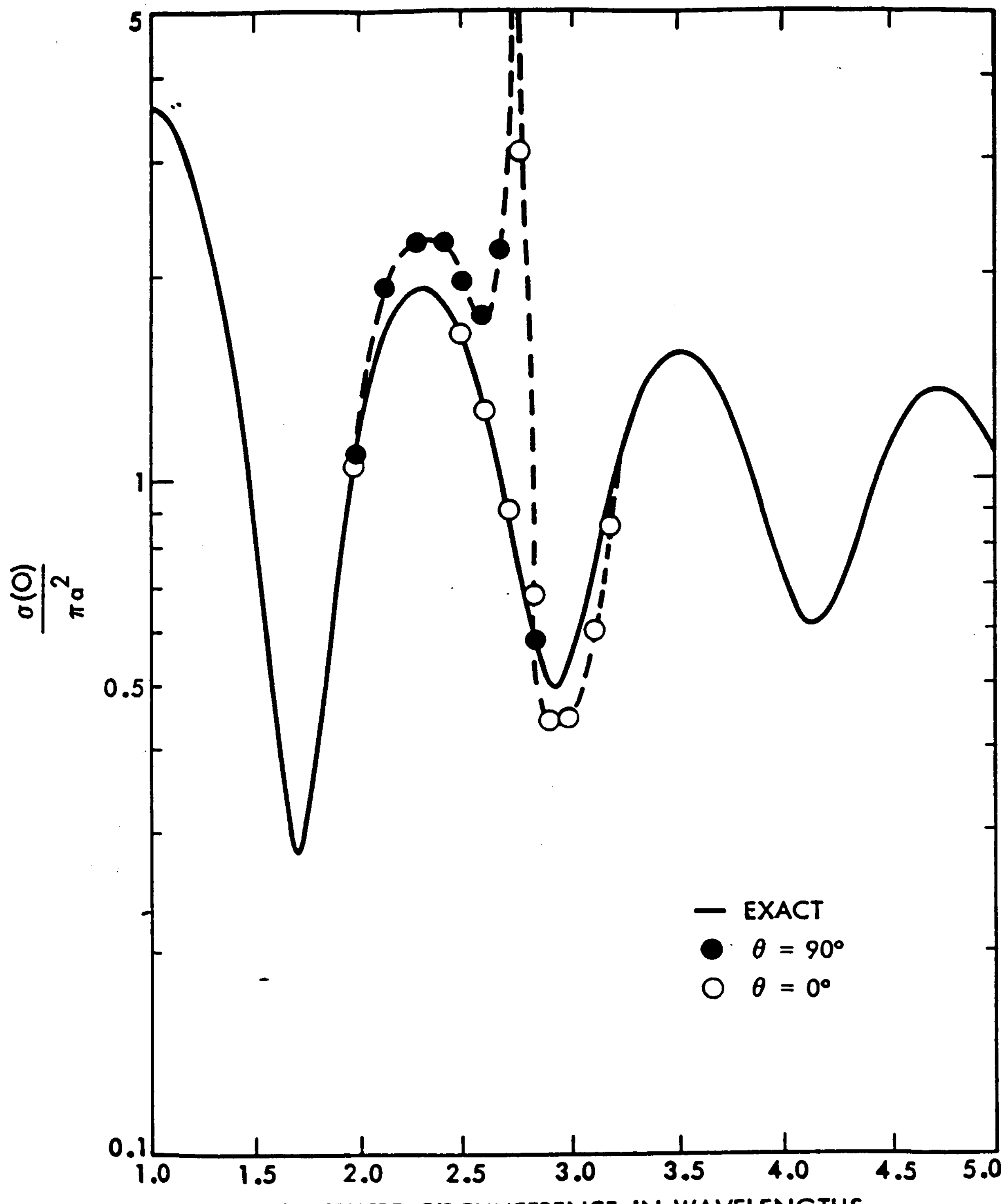


Figure 5.10 Comparison of NEC results with exact results for radar cross sections in the region of the cavity resonance. The filled dots are from NEC for the incident wave vector in the -x direction and the open dots are for the incident wave in the -z direction.

2.8 are plotted. The normalized radar cross section at ka=2.75 was 10.1 (off the graph) which is 14 times the exact value. For ka=2.6 and 2.7, the error was also on the order of 20%. For ka=2.8 (also 2.9 and 3.0), however, there is no observable error.

The other set of radar cross section points plotted in Figure 5.10 are for the same test problem, but with the wave incident from the -z direction (open dots). (The radar cross section, of course, is now calculated from the backscattered radiation in the +z direction.) If the patch model were perfectly spherically symmetric, the results for these two different angles of incidence should be identical. However, from Figure 5.2, it can be seen that the patch model used does not have perfect spherical symmetry. This apparently causes an asymmetry in the results. For radiation incident in the -z direction, the internal fields are apparently excited for values of ka at and slightly ABOVE the resonance at ka=2.744; in the first case (-x axis) the errors occurred for values at and slightly BELOW the resonance. For -z incidence, the values at ka=2.9,3.0 and 3.1 are now on the order of 20% below the exact value. The value at ka=2.7 has no observable error for this case.

The presence of large internal resonant fields can be detected by looking at the near fields inside the object. Such errors can be avoided by proper placement of wires inside the object to destroy the cavity resonances [5-1].

2. Sources of Error in PFDTD

There are two possible sources of error in the PFDTD results. First, the "staircase" approximation to the sphere results in a variation for the sphere radius of several percent. (For rectangular scatterers this error is zero.) Second, the PFDTD code has not been adequately validated due to lack of time. In particular, we need to check the symmetry of the results by running the code for plane waves incident from various directions.

From the CPU times we also conclude that PFDTD is considerably less efficient than NEC in solving for the scattered fields of the sphere. This is simply due to the generality of the FDTD method in comparison with NEC which is designed specifically for steady-state analysis of perfectly conducting scatterers. FDTD, on the other hand, can do either a steady-state or transient analysis of either perfectly conducting scatterers or volumetrically complex scatterers (e.g., inhomogeneous dielectrics). In all cases the PFDTD computation times would be comparable to or less than those of the test case.

Clearly, NEC and PFDTD are complementary codes, and, as this work demonstrates, the proper code must be used for a given situation if accurate results are to be efficiently obtained.

REFERENCES

[5-1] Burke, G. J. and Poggio, A. J., "Numerical Electromagnetics Code (NEC) - Method of Moments," (Lawrence Livermore National Laboratory, 1980).

[5-2] Ruck, Barrick, Stuart, Krickbaum, "Radar Cross Section Handbook," Vol. 1 (Plenum Press, 1970).

[5-3] Bowran, Senior, Uslenghi, "Electromagnetic and Acoustic Scattering by Simple Shapes," (Amsterdam, 1969).

SECTION VI

CONCLUSIONS

The basic objective of this task was to investigate the applicability of a parallel computing architecture to the solution of large scale electromagnetic scattering problems. To accomplish this objective, two scattering codes were implemented on JPL's parallel computer, the Mark III Hypercube. One code was the widely used NEC-2 method of moments code, obtained from Lawrence Livermore National Laboratory as a running VAX code. The parallel implementation and results of this code were described in Section IV. The second code implemented was a finite difference time domain code in which Maxwell's equations were solved explicitly in three dimensions as an initial value problem. No suitable existing version of such an approach was available and considerable code development was necessary before the parallel implementation of the finite difference time domain (FDTD) code could begin. THE FDTD code development, implementation and results were discussed in Section III. In Section V., the accuracy of the codes was checked by comparing results from both codes with each other and with exact results for the sponsor- selected test case, scattering from a perfectly conducting sphere. Very good agreement was found whenever code results were expected to be accurate.

There are three quantitative measures of performance of the applicability of parallel architecture to large scale electromagnetic scattering problems. The first two relate to performance relative to a conventional mainframe, here chosen to be a VAX 11/750: (1) What is the increase in the SPEED relative to a standard mainframe? and (2) What is the increase in the SIZE of the problem which can be run relative to a standard mainframe? The answers to these two questions depend on the particular computers used for the comparisons. The third measure relates more generally to whether or not the methods of solution and numerical algorithms are suitable for parallel architecture: How much faster is a problem run on multiple processors in a parallel computer compared to the same problem run on only one processor? Although the answer here will depend on the specifics of the parallel computer used, it is still one useful measure of the applicability of parallel computing architecture to such problems.

(1) Code Speed for 32 Node Mark III Relative to a Conventional Mainframe

Timing comparisons between the VAX and the Mark III 32 node for the NEC code were done for two different types of problems. One was a problem in which the object was modelled with patches (Table 4.1) and one was a problem in which the object was modelled with wires (Table 4.2). Comparisons were made for both because, for a fixed number of segments or patches, the matrix fill portion of the code takes much longer for wires. For the largest problems that could be run on the VAX, the Mark III 32-node was 7.5 times faster for the patch case and 11 times faster for the wire case. Thus a wire case that ran for an hour on the VAX took only 5.5 minutes on the Mark III Hypercube. This speedup makes it feasible for the user to run the NEC code in an interactive mode, rather than in a batch mode.

Direct timing comparisons between a VAX 11/750 and a Mark III Hypercube, using 32 active nodes, are not available for the finite difference time domain codes. Because the Parallel Finite Difference Time Domain (PFDTD) code has several enhancements, which we did not incorporate into the Generalized Finite Difference Time Domain (GFDTD) code, we cannot directly compare the execution times of these two distinct codes. The major difference in the two codes is the following: GFDTD uses first order correct radiation boundary conditions, while PFDTD uses the more computationally intensive second order correct radiation boundary conditions.

However, we compare the following execution times. We compare the execution time per iteration for GFDTD on a VAX 11/750 and a Counterpoint System XIX computer. The Counterpoint computer serves as the control processor for the Mark III Hypercube. For approximately 64000 cells in the global lattice, the execution time per iteration on the VAX is 47.47 sec. and the execution time per iteration on the Counterpoint is 89.81 sec. For 64000 cells, the execution time per iteration on the Mark III using 32 nodes is 1.70 sec. The ratio of the execution time on the VAX over the execution time on a Mark III using 32 nodes is 27.9. If GFDTD also had the enhanced features of PFDTD, this ratio would be larger. The reason for the larger ratio is the following: second order correct boundary conditions would increase the execution time per iteration on the VAX. Since a typical PFDTD run lasts about 1 to 2 hours, the same job would take over 27.9 to 55.8 hours on the VAX.

(2) Increase in Problem Code Size Relative to a Conventional Mainframe

In addition to providing multiple processors for increased computing speed, parallel computing architecture also offers the possibility of an increase in available memory if each processor has its own memory as well as its own CPU. The VAX 11/750 has about 5 Megabytes of memory to a typical user whereas the Mark III has 4 Megabytes per processor, or a total of 128 Megabytes. Clearly, much larger problems can be run on the Mark III 32 Node.

For the NEC code, the size of the largest problem that can be run is essentially determined by the largest interaction matrix which can be stored in memory. The matrix is N_T x N_T with $N_T = N + 2 * M$ where N is the number of wire segments and M is the number of patches used to model the object. The largest sphere case that could be run on the VAX was one with approximately 224 patches. On the Mark III 32 node, the largest patch problem run was one with 440 patches, or an 880 X 880 interaction matrix. This is roughly a factor of 2 in terms of the number of patches in the modeled object relative to the VAX.

For GFDTD and PFDTD, the largest size problem is determined by the number of unit cells used to model the computation lattice. For the VAX 11/750, we can allocate memory for about 192,000 unit cells. For the Mark III Hypercube with 32 active nodes, we can allocate memory for about 2,048,000 unit cells.

(3) Code Speed on 32 Nodes Relative to 1 Node

To measure the suitability of the codes to the parallel architecture of the hypercube, we use the speedup factor, defined as the ratio of the run time on n-nodes to the time on 1 node. If there were no penalties associated with parallel computing architecture, the speedup would be n. However, when a code is run on multiple nodes of a parallel processor, the speedup never increases linearly with the number of processors because there is "overhead" associated with running in parallel. The parallel overhead has two sources: time spent in communication between processors and time lost if the work loads of the various processors are not balanced so that some processors must sit idle while waiting for other processors to finish. Some algorithms are by nature more suited to parallel computers than others.

The GFTDT code is an ideal candidate for parallel architecture because the finite difference method used is intrinsically a "local" calculation. The computation at each grid point utilizes information from, at most, two unit cells. This local calculation feature permits low parallel communication overhead. Also, load imbalance is not a serious problem because we can evenly divide the global grid among the nodes. The suitability of this code for parallel computing is reflected in the high speedup factors. For the largest problem that could be run on 1 node, the speedup factor in going from 1 to 32 nodes was 25.4 (or 79.3% efficient). It is important to note that because the problem size remains the same, the number of cells in each processor of a 32 node configuration is reduced by a factor of 32. The larger hypercube configurations do not work to full capacity. In other words, the ratio of computation to communication times is reduced. For a larger problem, where we run the same size problem on 32 nodes that saturates an 8 node configuration, the number of cells in each processor is reduced by a factor of 4. The speedup factor in going from 8 to 32 nodes is 3.7 (or 92.7% efficient).

For the NEC code, speedups were given for the two computationally intensive parts of the code separately, the matrix fill and the Householder transformation of the matrix (factorization time). The speedup in total times was also given. The fill of the interaction matrix is also ideally suited to parallel architecture because the various rows of the matrix can be filled independently of each other, leading to a very low communication overhead. However, when the number of rows is not much larger than the number of processors, load imbalance problems do occur. Nevertheless, the speedups found for the fill were excellent. For the largest problem that could fit in one node (240 wire segments), the speedup factor for 32 nodes relative to 1 node was 26.8.

Because by nature matrix operations generally need information from many or all of the other matrix elements, matrix computations typically demonstrate less speedup than other algorithms, where the data internode communication requirements are low or confined to neighboring processors. For this reason concurrent implementation of matrix algorithms requires great care. Data must be distributed so that the amount of communication is minimized. For parallel NEC, the internode data dependency is reflected in the relatively lower speedups for the factorization part compared to the fill. As shown in Figures 4.21 and 4.24, appreciable speedups of 23 to 24 times have been obtained particularly for larger problems.

This task has demonstrated the applicability of a parallel architecture to the solution of large electromagnetic scattering problems. The two techniques used, finite difference and method of moments, have provided insight into the comparative speedups which can be attained for several algorithms. It also has demonstrated the flexibility of the hypercube architecture for different applications. The 32-node Mark III Hypercube has been used to concurrently solve electromagnetic scattering problems too large and/or too time consuming to be done on a sequential computer such as the VAX. The ability to measure this performance for variable size hypercubes allows us to extrapolate the performance of even larger hypercube configurations on larger problems.

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No. JPL Publication 87-18	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Hypercube Matrix Computation Task - Final Report for 1986-87		5. Report Date July 1, 1987
		6. Performing Organization Code
7. Author(s) R. Calalo, W. Imbriale, P. Liewer, J. Lyons, F. Manshadi, and J. Patterson		8. Performing Organization Report No.
9. Performing Organization Name and Address JET PROPULSION LABORATORY California Institute of Technology 4800 Oak Grove Drive Pasadena, California 91109		10. Work Unit No.
		11. Contract or Grant No. NAS7-918
		13. Type of Report and Period Covered JPL Publication
12. Sponsoring Agency Name and Address NATIONAL AERONAUTICS AND SPACE ADMINISTRATION Washington, D.C. 20546		14. Sponsoring Agency Code RE232 PX-644-11-00-00-08
15. Supplementary Notes		

16. Abstract

The objective of the Hypercube Matrix Computation (Year 1986-1987) task was to investigate the applicability of a parallel computing architecture to the solution of large scale electromagnetic scattering problems. Two existing electromagnetic scattering codes were selected for conversion to the Mark III Hypercube concurrent computing environment. These two codes were selected so that the underlying numerical algorithms utilized would be different thereby providing a more thorough evaluation of the appropriateness of the parallel environment for these types of problems. The first code was a frequency domain method of moments solution, NEC-2, developed at Lawrence Livermore National Laboratory. The seond code was a time domain finite difference solution of Maxwell's equations to solve for the scattered fields.

Once the codes were implemented on the hypercube and verified to obtain correct solutions by comparing the results with those from sequential runs, several measures were used to evaluate the performance of the two codes. First, a comparison was provided of the problem size possible on the hypercube with 128 megabytes of memory for a 32-node configuration with that available in a typical sequential user environment of 4 to 8 megabytes. Then, the performance of the codes was analyzed for the computational speedup attained by the parallel architecture.

17. Key Words (Selected by Author(s)) Electronics and Electrical Engineering; Computer Systems; Numerical Analysis; Electricity and Magnetism	18. Distribution Statement Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	120	